MANUAL FOR SCHOONSCHIP

A CDC 6000/7000 program for symbolic evaluation of algebraic expressions

(Submitted to Computer Physics Communications)
SCHOONSCHIP is a high-speed program which is capable of evaluating expressions of the form:

\[(A_1B_1+\ldots)\times(A_2B_2+\ldots)\times\ldots + (A_{A_1}B_{B_1}+\ldots)\times\ldots\]

where \* denotes multiplication and where \(A_1, B_1, \ldots\) may be products of numbers, algebraic symbols, vectors, functions, etc., or further expressions enclosed in brackets.

Moreover, an elaborate set of substitutions and commands is provided which allows one to perform most of the commonly required algebraic manipulations. Many special operations which are used in high-energy physics calculations, are built in as well.

Its results can be given again as input for further processing, hence using the computer as a writing pad. Output, compatible with FORTRAN, can be obtained on punched cards and inserted directly in a numerical program.
APPENDIX 1: Example. Expansion in power series
APPENDIX 2: Example. A calculation from high energy physics
APPENDIX 3: Index of SCHOONSCHIP statements (ordered by action)
APPENDIX 4: Index of SCHOONSCHIP statements (in alphabetical order)
FOREWORD

SCHOONSCHIP was first written in the early 1960's by M. Veltman. His CERN preprint, "SCHOONSCHIP, a CDC 6600 program for symbolic evaluation of algebraic expressions", July 1967, describes the features of the 1967 version. Since that time, SCHOONSCHIP has evolved to such an extent that it became necessary to completely revise the old manual. This new manual describes the 1973 version.

Clearly, the evolution of SCHOONSCHIP will go on and this manual will soon be incomplete or even wrong on some points. However, there exists the program YNGLING which is rigorously kept up to date and is distributed together with the SCHOONSCHIP source code. This program is intended to teach a new user, step by step, how to write his SCHOONSCHIP programs. It consists of many examples and comments (approximately 100 pages) and illustrates all the available instructions. Moreover, it gives some information on the internal working of SCHOONSCHIP.

Any CDC 6000/7000 computer installation can request SCHOONSCHIP from the CERN Program Library (where it is catalogued under the number R201) by mailing them a (400 ft) tape and specifying their computer and SCOPE system. The SCHOONSCHIP source code contains approximately 40,000 assembly statements and is consequently completely machine dependent. It can only run on CDC 6000 or CDC 7000 machines and requires approximately 25,000 (decimal) words of central memory.

1. BASIC OPERATION OF SCHOONSCHIP

1.1 Manipulation of algebraic symbols

The operation of the program will be explained by looking at some specific, very simple examples. Consider

\[(A + B)(A + B)\].

The program accepts expressions of this type and works out the brackets. Thus it will give, as a first result:

\[A^2 + A*B + B*A + B^2\].

(* = multiplication, ** = exponentiation). However, internally a fixed order (depending on the names of the algebraic symbols) is used for factors in a term and we get

\[A^2 + A*B + A*B + B^2\].

As a last step, while storing the result, all terms are mutually compared, and if possible their numerical coefficients are added. Thus we get

\[A^2 + 2*A*B + B^2\].

This is the basic function of the program. As one observes, some rules pertinent to the particular kind of symbols involved are built in:
i) \( A^{**1}A^{**1} = A^{**2} \)

ii) \( A*B = B*A \).

Other quantities that follow different rules may be used also and are described further on (for instance, vectors, Dirac matrices, etc.).

A very important facility is the substitution facility. This enables one to make changes or substitutions in the course of the calculation, or to select a particular type of term. To illustrate this, suppose that we want to evaluate the following integral:

\[
\int (x+5*A*x)\cdot(x+2+3*B*x) \, dx.
\]

In such a case we know beforehand what type of integrals may appear and we could make a list:

\[
\begin{align*}
A^{**3} &= x^{**4}/4 \\
A^{**2} &= x^{**3}/3 \\
A^{**1} &= x^{**2}/2 .
\end{align*}
\]

The program accepts this set of substitutions and will compare each produced term with the given table and make the substitutions indicated. After that the result is sorted as before.

More generally, a SCHOONSCHIP program consists of an initial expression (called Z expression) and a set of substitutions to be applied to that expression. The calculation mechanism then works roughly as follows:

The first term of the Z expression is fetched from storage. A scan through the set of substitutions is made. If one of them applies to that (so-called) incoming term, the term is modified accordingly. Then the term is stored again, and the next term is fetched from storage. The same scan is repeated etc., until all terms are dealt with.

Calculations often have a more complex structure: we can require a new set of substitutions to be applied to the expression which resulted after performing the previous set of substitutions. In our example, that second set of substitutions is: \( x = A - B \).

This facility is easily incorporated, by restarting the previous mechanism:

Fetch the first term of the (in the meantime modified) Z expression.

Apply the \( n^{th} \) set of substitutions and store the modified term, and so on until all terms are dealt with. Increment the set number to \( n + 1 \).

Repeat this procedure till all sets of substitutions are applied.
When everything is done, terms are sorted and printed. In other words, addition of terms is not performed after each set of substitutions.

It is up to the programmer to arrange his substitutions into different sets. He has to assign a LEVEL to each of his substitutions: 1, 2, 3, ... . The level is a kind of label for the substitution sets. During the calculation, SCHOONSCHIP keeps track of the execution level in such a way that every time an elementary process, which
- fetches from storage
- inspects for substitutions, assigned to that level
- restores

is finished, the level is increased by 1.

In the simplest cases, this level is identical with the above-mentioned set number. Differences arise when more complicated substitutions are used which require several levels (i.e. several of these elementary processes) for their execution. This happens for instance when the right-hand side of a substitution is an expression which contains brackets that have to be worked out.

As a final remark, it has to be pointed out that the result of a calculation is often so big that it would flood the computer memory. In such cases SCHOONSCHIP automatically uses a mechanism to overflow onto disk.

1.2 Properties of functions, vectors and spinors

We established in the previous paragraph the basic rules for algebraic symbols:

\[ ab = ba \]
\[ a^m a^n = a^{m+n}, \]

where m and n are numbers. Rules pertinent to vectors, dot products and functions will now be described. At the same time a summary of the built-in Spinor Algebra will be given.

Non-commuting quantities are always called functions, whether they depend explicitly on other variables or not. As a specific example we mention the Dirac matrices, to be described below.

Some of our rules are metric-dependent (especially the rule \( \delta_{\mu\nu} = r \)) where r = dimension of the space involved), and the metric intended is such that the metric tensor is +1 on the diagonal and zero otherwise. This corresponds to

\[ p^2 = p^1 p^1 + p^2 p^2 + p^3 p^3 + p^4 p^4 = -m^2 \]

for a (relativistic) particle of momentum p and mass m.
A vector in $r$-dimensional space may be denoted by $p_\mu$ where the index $\mu$ goes from 1 to $r$. We assume that a dot product is defined, i.e. with any two vectors $p$ and $q$ there may be associated a dot product $(pq)$ that satisfies the rule $(pq) = (qp)$. This dot product behaves like a commuting algebraic symbol, e.g. $(pq)^m (qp)^n = (qp)^{m+n}$, where the exponents $m$ and $n$ are numbers.

The formal relation between vectors and their dot product is $p_\mu q_\mu \equiv (pq)$. This coincides with the usual definition of a dot product if the summation convention is adopted (double-occurring indices imply a summation: $p_\mu q_\mu$ stands for $\sum p_\mu q_\mu$). In other words, SCHOONSCHIP does not know the relation $(pq) = p_1 q_1 + p_2 q_2 + \cdots + p_r q_r$.

Here, $p_1$, $p_2$, ... are specific components of vector $p$. They behave differently from $p_\mu$:

- **i)** Both arguments are numbers:
  
  \[
  (pq) = (qp)
  \]

- **ii)** One argument is a number; the other an index:

  \[
  p_1 p_1 = p_1^2 \quad (m, n \text{ are numbers})
  \]

- **iii)** Both arguments are indices:

  \[
  \delta_{\mu \nu} = 1 \quad \text{if } \mu = \nu , \quad \delta_{\mu \nu} = 0 \quad \text{if } \mu \neq \nu .
  \]

Its effect on, for example, $p_\mu \delta_{\mu \nu}$ is clear when the implied summation is performed:

\[
p_\mu \delta_{\mu \nu} = p_1 \delta_{1 \nu} + p_2 \delta_{2 \nu} + p_3 \delta_{3 \nu} + \cdots = p_\nu .
\]

As pointed out before, SCHOONSCHIP does not perform such a summation. However, exactly the same effect can be obtained by defining a function $D$ as follows (assume $m$, $n$ are numbers, $\mu$, $\nu$ are indices, $p$, $q$ are vectors and $F$ is a function):

**i)** Both arguments are numbers:

\[
D_{mn} = 1 \quad \text{if } m = n , \quad \text{e.g. } D_{33} = 1
\]

\[
D_{mn} = 0 \quad \text{if } m \neq n , \quad \text{e.g. } D_{32} = 0 .
\]

**ii)** One argument is a number; the other an index:

$D_{\mu m}$ replaces all $\mu$'s by the value $m$, e.g. $p_\mu D_{\mu 3} = p_3$.

**iii)** Both arguments are indices:

$D_{\mu \nu}$ replaces all $\mu$'s by $\nu$.

- **e.g.** $F(\mu) F(\nu) D_{\mu \nu} = F(\nu) F(\nu)\]

\[
p_\mu q_\nu D_{\mu \nu} = (pq)
\]

\[
D_{\mu \nu} D_{\nu \lambda} = D_{\mu \lambda} .
\]
Finally, the rules \( D_{\mu \nu} = r (r = \text{dimension of the index } \mu) \) and \( D_{\mu^* \nu} = -D_{\mu \nu} = D_{\mu \nu} \) are assumed. This last property is consistent with another convention: \( p_{-\mu} = -p_{\mu} \).

As a further development in notation, we introduce the rule that a vector may occur at any place where an index is allowed to occur. The meaning of the symbols so introduced is as follows:

- \( p_{\mu \nu} = (pq) \)
- \( D_{\mu \nu} = p_{\mu} \)
- \( D_{\mu \nu} = (pq) \)
- \( D_{\mu \nu} = p_{\nu} \)

This notation sometimes avoids the necessity of introducing twice-occurring indices.

The totally antisymmetric tensor \( \epsilon_{\mu_1 \mu_2 \ldots \mu_n} \) of dimension \( n \) is built in. As an example, the properties of the four-dimensional tensor are given:

- \( \epsilon_{\mu \nu \alpha \beta} \) is antisymmetric under the exchange of any two indices, e.g. \( \epsilon_{\mu \nu \alpha \beta} = -\epsilon_{\nu \mu \alpha \beta} \).

Further,

\[
\begin{align*}
\epsilon_{\mu \nu \alpha \beta} \epsilon_{\mu \nu \alpha \beta} &= 24 \\
\epsilon_{\mu \nu \alpha \beta} \epsilon_{\mu \nu \alpha \lambda} &= 6 \delta_{\beta \lambda} \\
\epsilon_{\mu \nu \alpha \beta} \epsilon_{\mu \nu \lambda \kappa} &= 2 (\delta_{\alpha \lambda} \delta_{\beta \kappa} - \delta_{\alpha \kappa} \delta_{\beta \lambda}) = 2 \left| \begin{array}{rr}
\delta_{\alpha \lambda} & \delta_{\beta \lambda} \\
\delta_{\alpha \kappa} & \delta_{\beta \kappa}
\end{array} \right|
\end{align*}
\]

and analogously with coefficient 1 in front of the \( 4 \times 4 \) determinant if no indices are equal.

In this way a product of \( \epsilon \)'s may always be reduced to zero or one \( \epsilon \). This reduction is applied when the command TRICK is given.

Finally we discuss briefly \( \gamma \)-algebra. Consider the four Dirac matrices \( \gamma^1, \gamma^2, \gamma^3 \) and \( \gamma^4 \) and the identity 1 obeying the rules:

\[
\gamma^\mu \gamma^\nu = -\gamma^\nu \gamma^\mu + 2 \delta_{\mu \nu} \cdot 1.
\]

It is convenient to define the following quantities:

\[
\begin{align*}
\gamma^5 &= \gamma^1 \gamma^2 \gamma^3 \gamma^4, \\
\gamma^6 &= 1 + \gamma^5, \\
\gamma^7 &= 1 - \gamma^5.
\end{align*}
\]
One finds
\[ \gamma^5 \gamma^\mu = -\gamma^\mu \gamma^5 \]
\[ \gamma^5 \gamma^5 = 1. \]

It is always possible to reduce a product of \( \gamma \)-matrices to a sum of terms each with at most 2 \( \gamma \)'s in it.

The basic reduction rule is
\[ \gamma^\mu \gamma^\nu \gamma^\alpha = \delta_{\mu \nu} \gamma^\alpha + \delta_{\mu \alpha} \gamma^\nu + \delta_{\nu \alpha} \gamma^\mu + \epsilon_{\mu \nu \alpha \beta} \gamma^\beta. \]

For more complicated cases, some special algorithms exist which are much more efficient than applying the basic reduction rule over and over again; for instance the Kahane algorithm, which contains the Chisholm identities as a special case.

[N.B. This algorithm is only valid in 4-dimensional space.] We state now the Chisholm identities:

Let \( S \) be a product of a certain number of \( \gamma \)'s, and \( S^R \) be the same product, but with the \( \gamma \)'s in opposite order. Thus, if \( S = \gamma^\alpha \gamma^\beta \gamma^\mu \) then \( S^R = \gamma^\mu \gamma^\beta \gamma^\alpha \).

Furthermore brackets preceded by \( \text{Tr} \) indicate that the trace must be taken according to the rules to be given below. Finally \( \gamma^\alpha p_\alpha = \gamma^\alpha = p \), where \( p \) is a vector. We will call \( S \) odd (even) if \( S \) contains an odd (even) number of \( \gamma \)'s. We have
\[ (p^2 = \text{dot product of } p \text{ with itself}): \]
\[ \gamma^\mu S_\mu^\nu = -2 S^R_\nu \quad \text{if } S \text{ odd} \]
\[ \gamma^\mu S^R_\nu = \text{Tr} (S^R_\mu) \cdot 1 - \gamma^5 \text{Tr} (S^5_\nu S^R_\mu) \quad \text{if } S \text{ even} \]
\[ \psi S \psi = -p^2 S^R_\nu + \frac{1}{2} \psi \text{Tr} (S^R_\nu) + \frac{1}{2} \psi \gamma^5 \text{Tr} (\gamma^5 S^R_\nu) \quad \text{if } S \text{ odd} \]
\[ \psi S \psi = -p^2 S^R_\nu + \frac{1}{2} \gamma^5 \psi \text{Tr} (\gamma^5 S^R_\nu) \quad \text{if } S \text{ even}. \]

The Kahane algorithm reduces expressions of the form \( \gamma^S_1 \gamma^S_2 \gamma^S_3 \gamma^S \ldots \). Upon the command \text{TRICK}, \text{SCHOONSCHIP} does the reduction of the \( \gamma \)'s. It decides which of the available identities should be applied.

The trace of a product of \( \gamma \)'s is given through the above-quoted reduction rule (reducing any product of \( \gamma \)'s to at most 2 \( \gamma \)'s with or without a \( \gamma^5 \)) in combination with the rules:

\[ \text{Tr} \; (1) = 4 \]
\[ \text{Tr} \; (\gamma^1) = \text{Tr} \; (\gamma^5) = 0 \]
\[ \text{Tr} \; (\gamma^\mu \gamma^\nu) = 4 \delta_{\mu \nu} \]
\[ \text{Tr} \; (\gamma^\mu \gamma^5) = \text{Tr} \; (\gamma^5 \gamma^\mu) = 0 \]
\[ \text{Tr} \; (\gamma^5 \gamma^\mu \gamma^\nu) = \text{Tr} \; (\gamma^\mu \gamma^\nu \gamma^5) = \text{Tr} \; (\gamma^\mu \gamma^\nu \gamma^5) = 0. \]

The trace of a product of \( \gamma \)'s is computed according to these rules upon the command \text{TRACE}. 

As an additional feature \( \gamma \)-symbols will have another index, the so-called loop index, to denote different sets of \( \gamma \)-symbols. The rules given above are pertinent to \( \gamma \)'s within a set. \( \gamma \)'s of different sets behave with respect to each other as ordinary commuting algebraic symbols.

In connection with \( \gamma \)-symbols we have the spinor symbols \( u \) and \( \bar{u} \). These spinor symbols are related to particles of given spin, mass and momentum. The program distinguishes between spinors for spin \( \frac{1}{2} \) and spinors for spin \( \frac{3}{2} \) particles. For a product of a spinor \( u \) and a spinor \( \bar{u} \) of one and the same particle one may define the operation "spin sum" by (\( k \) and \( j \) are loop indices)

\[
u(k,m,p) \bar{u}(j,m,p) = -i\phi + m\cdot l \quad \text{for a spin } \frac{1}{2} \text{ particle,}
\]

where the \( \gamma \) in \( \phi \) and \( l \) are given the loop index \( k \). For the other \( \gamma \)'s and spinors occurring anywhere else in the product the loop index \( j \) is changed to \( k \). The order of the terms in the product is changed: in reading the result after spin summation the program jumps from \( u \) to \( \bar{u} \) and comes back later to collect possible factors in between. Thus, with \( r, j \) and \( k \) being loop indices the spin-sum operation changes

\[
\gamma^\alpha_r \gamma^\beta_r u(r,m,p) \gamma^\mu_j \gamma^\nu_j \bar{u}(j,m,p) \gamma^\lambda_j \gamma^\sigma_j
\]

into

\[
\gamma^\alpha_r \gamma^\beta_r (-i\phi + m\cdot l) \gamma^\lambda_j \gamma^\sigma_j \gamma^\mu_k \gamma^\nu_k.
\]

In requesting spin summation with the command SPIN1 one must specify a loop index (\( r \) in the above example) and the program searches then for a spinor \( u \) with that loop index, and a spinor \( \bar{u} \) with the same mass and momentum variable. If found the above manipulation is performed. After this the program starts again looking for a spinor \( u \) with loop index \( r \), etc. This goes on till no such spinor \( u \) is found anymore.

For spin \( \frac{3}{2} \) particles the spinor \( u \) and \( \bar{u} \) carry an extra index and the built-in rule is

\[
u(k,\mu,m,p) \bar{u}(j,\nu,m,p) = \left\{ \begin{array}{l} 1 \cdot \delta_{\mu\nu} - \frac{1}{3} \gamma^\mu \gamma^\nu - \frac{i}{3m}(\gamma^\mu p_\nu - p_\mu \gamma^\nu) + \frac{2}{3m^2} 1 \cdot p_\mu p_\nu \end{array} \right\} \times
\]

\[
\left\{ -i\phi + m\cdot l \right\},
\]

where \( l \) and all \( \gamma \)-symbols have the loop index \( k \).

We close this brief review by noting that an antiparticle spinor differs from a particle spinor by the mass \( m \) being replaced by \( -m \).
2. BASIC QUANTITIES: NOTATION AND DEFINITION

The following kinds of quantities are possible in the input:

a) numerical quantities,
b) algebraic symbols,
c) indices,
d) functions,
e) vectors,
f) dot products,
g) specific components of a vector,
h) operators,
i) dummies,
j) expressions enclosed in brackets.

2.1 Numerical quantities

Numerical quantities may occur as a factor in a product, as an exponent of some quantity, as an index of a vector to denote a specific component, and as an argument of a function.

As a factor in a product a numerical quantity may be written with or without a decimal point and with or without a signed exponent which may also be with or without a decimal point. The exponent is indicated through the letter E and denotes the power of 10 by which the number has to be multiplied.

Examples: The number 831 may be written as

\[ \begin{align*}
831 & \quad 831.0 \quad 83.1E + 1 \\
83.10.0E - 1.0 & \quad 0.831E3 \quad 83.1E + 1.0
\end{align*} \]

A number should never be followed by **, the symbol for exponentiation. Thus 2.4**3 is (although in principle well defined) unacceptable. But 2.4*8.3 is valid, * implying multiplication. Also 22.4/9.3E + 7 is allowed, where / implies division.

In all other situations numerical quantities must be signed or unsigned integers less than 128 in magnitude. As vector indices they must be less than 32, and unsigned.

Examples:

\[ \begin{align*}
A^{*107} & \quad P(4) \quad F(-3,A,P) \\
A^{*- 107} & \quad A^{*(-107)}
\end{align*} \]

where A, P and F denote an algebraic symbol, a vector and a function, respectively.

Exception: The index of the built-in function DF can go as high as 4096.
2.2 Algebraic symbols

Algebraic symbols are denoted by a string of one to five alphanumeric characters, the first being non-numeric, possibly followed by an exponent which can be enclosed in brackets. If no exponent is present it is assumed to be +1. An algebraic symbol may occur as a factor in a product, or as an argument of a function. If the symbol is immediately preceded by a /, the sign of the exponent is changed and the operation of division is replaced by multiplication.

Examples:

A  A1B**2  AB**2 .

With respect to the / sign we have:

1/A  = A**-1
1/AB**2  = AB**2
1/A*A1B**2  = A**-1*A1B**2
1/A/A1B**2  = A**-1*A1B**2 .

A variable is defined to be an algebraic symbol if it is declared in the S-list (symbol list; see later). At most 215 different symbols are allowed. The symbol I, satisfying the rule I**2 = -1 is built in.

2.3 Indices

Indices are denoted by a string of one to five alphanumeric characters, the first being non-numeric. They may occur as vector indices, as function arguments or as factors in a product. In the last case they may be followed by an exponent, and are treated as algebraic symbols.

Examples:

P(MU3)  J**2  1/L12J**3  F(MU3,NU,P)

where F and P denote a function and a vector, respectively. MU3, J, L12J and NU are indices.

A variable is defined to be an index if it occurs in the I-list (index list). If it also occurs in the SUM list, it is a summation index. At most 250 different indices are allowed.

2.4 Functions

Functions are denoted by a string of one to five alphanumeric characters, the first being non-numeric. They may be followed by arguments separated by comma's and enclosed in brackets. A function may appear as a factor in a product or as an argument of a function. A function must never be followed by an exponent. Rules and possibilities concerning the arguments of a function will be summarized in Section 2.11.
Examples:

\[ F_1 \cdot F_2(A,B,C) \quad F_3(F_2,F_1,A,B,C) \]

where \( A, B, C \) are algebraic symbols and \( F_1, F_2, F_3 \) are functions.

A variable is defined to be a function if it occurs in the \( F \)-list (function list). At most 200 different functions are allowed.

2.5 Vectors

Vectors are denoted by one or two alphanumeric characters, the first being non-numeric. A vector may occur as a factor in a product, in which case it is followed by an index, vector or number enclosed in brackets (the last two cases really are given in Sections 2.6 and 2.7, see below), or it may occur as an argument of a function or of another vector. A vector must never be followed by an exponent. The rule \( P(-X) = -P(X) \) with \( X \) index or vector is built in.

Examples:

\[ P(MU) \quad P_1(ALPHA) \quad F(P,P_1) \quad P(-Q) \quad P_1(4) \]

where \( P, P_1 \) and \( Q \) are vectors, \( MU \) and \( ALPHA \) indices, and \( F \) is a function. A variable is defined to be a vector if it occurs in the \( V \)-list (vector list). At most 30 different vectors are allowed.

2.6 Dot products

Dot products are denoted by a name consisting of two vector names separated by the letter \( D \). They may occur as a factor in a product, in which case they may be followed by an exponent and are treated as algebraic symbols. They may also appear as an argument of a function.

Examples:

\[ P_1D_1P \quad P_2D_2P \quad 1/P_3D_3P \quad F(PDQ) \]

A variable is defined to be a dot product if its name contains the letter \( D \) with on both sides a quantity occurring in the \( V \)-list.

2.7 Specific components of a vector

Specific components of a vector are denoted by a vector name followed by a positive number less than 32 enclosed in brackets. They may occur as a factor in a product, in which case they may be followed by an exponent and are treated as algebraic symbols. They may also appear as an argument of a function.

Examples:

\[ P(4) \quad P_1(8) \quad P(4)**3 \quad 1/P_1(8)**3 \quad F(P(4)) \]
A variable is defined to be a specific component of a vector if its name consists of a quantity occurring in the V-list followed by a number enclosed in brackets.

2.8 Operators

An operator is a quantity occurring in the internal operator list followed by an expression enclosed in brackets on which the operation is to be performed.

Examples:

\[
\text{INTEG} \left( (3.3)^2 - (2.2)^2 \right)
\]

\[
\text{CONJG} \left( G(J1,P) + M \right)
\]

The operator CONJG takes the complex conjugation. The operator INTEG does the numerical evaluation of an expression and converts the result to an integer.

2.9 Dummies

Dummies are quantities that are used in case an expression must be evaluated several times with different arguments. A function may then be defined once, with dummy symbols at the places where the different arguments are to be used. A dummy must nevertheless be defined in an S-, V-, F- or I-list, depending on what kind of quantity the dummy represents. A dummy is defined as such by occurrence on the left-hand side of a substitution (see later) and is followed by a + sign.

Examples:

\[
\text{ALF}(J1,M+,P+) = 2*G(J1,P) + I*PDQ**3*M
\]

This means that the right-hand side must be inserted whenever the function ALF with the first variable being J1 is encountered in the initial expression. The second and third variable are taken over and put in the right-hand side instead of M and P. Although P and M are dummies, they must be given in the V- and S-lists, respectively, and the replacements during the calculation must be with vectors and algebraic symbols, respectively. If in the actual calculation an argument is some expression enclosed in brackets, the corresponding dummy must be an algebraic symbol. Thus with the above substitution, the quantity ALF(J1,(A+B),Q) is computed correctly.

Dummies to be replaced by numbers must be indices.

2.10 Expressions enclosed in brackets. Numerical expressions

Expressions enclosed in brackets may appear as factors in a product (possibly followed by a non-negative exponent) or as arguments of a function (never with an exponent). An expression consists of a number of terms, separated by + or - signs. Each term consists of a product of factors as given in Sections 2.1-2.10.
Moreover -0 is replaced by +0. Its numerical value is then calculated, converted to integer and treated modulo 128. If an expression is assumed to be numerical (if it is not, it is an illegal argument). If an expression -- not in brackets -- is encountered as a function argument, that expression is assumed to be numerical. (If it is not, it is an illegal argument). Its numerical value is then calculated, converted to integer and treated modulo 128. Moreover -0 is replaced by +0.

**Examples:**

\[(A*P(MU)*(C+C1) - 2*Q(4)*ALF(J1,(A+B),A))\]
\[((F(A,B,C))**7)\]
\[(3.0)\]

**Exception:**

A numerical expression can have a negative exponent. An expression is said to be numerical if it reduces to a number at the moment it has to be used. The numerical evaluation of an expression is always attempted before any algebraic calculation is done with it.

**Example:**

The substitution: \(A^{**N+B**M+} = (N-M)**2\) acting on the incoming term \(A^{**5*B**2}\).

This will lead to \((5-2)**2\), which becomes \((3)**2\). This is a purely numerical calculation. Any non-numerical expression to a negative power, however, requires a special facility to divide by polynomials. Such a facility is not built into SCHOONSCHIP.

**2.11 Summary about function arguments**

Function arguments may be any of the symbols given in Sections 2.1-2.10. These arguments may be preceded by a minus sign. In particular, the use of numerical arguments is repeated here.

\(F(2)\) and \(F((17.3))\) are accepted in the way they stand here. \(F(2.2)\) and \(F(317)\) are accepted, but truncated to an integer and treated modulo 128 (without warning message). In other words, the 8 least significant bits of the resulting integer are taken and interpreted as one sign bit and a 7-bit number. In this way the quantity \(-0\) can be created:

\(F(255) = F(-0);\quad F(256) = F(+0)\).

An expression enclosed in brackets as a function argument may be preceded by an operator and a minus sign.

**Example:**

\(ALF(INTEG(N+M),-CONJG(A-I*C),-(U*V))\).

If an expression -- not in brackets -- is encountered as a function argument, that expression is assumed to be numerical. (If it is not, it is an illegal argument). Its numerical value is then calculated, converted to integer and treated modulo 128. Moreover \(-0\) is replaced by \(+0\).
Example:

\[ \text{ALF}(N+) = \text{ALF}(N-1)/(N-1) \]

Internally, \( \text{ALF}(N-1) \) is transformed at read-in time to \( \text{ALF}(\text{INTEG}(N-1)) \).

Finally, in connection with the command COMPO (see later), the quantities \( =A \) and \( =B \) and ... and \( =Z \) and \( * \) are also accepted as legal function arguments.

3. STRUCTURE OF A SCHOONSCHIP PROGRAM

Schematically, a SCHOONSCHIP program consists of one or more sections, each one built up with some of the following parts (in this order):

i) Declarations of the type of the used variables.
ii) Definitions, which create special functions.
iii) The expressions to be evaluated.
iv) The substitutions and commands to be applied to those expressions.
v) Various specifications about the format of the printed output.
vi) Instructions to store the results (on disk, on punched cards or in memory) after they have been printed.
vii) The execution instruction.

These parts will now be described in a more convenient sequence.

3.1 Expressions to be evaluated. Z and R expressions

The expressions to be evaluated have to be given in the following format:

\[ Z \text{ NAME} (\text{arg1, arg2,} \ldots) = \text{expression} \]

or

\[ Z \text{ NAME} (\text{IND, arg1, arg2,} \ldots) = \text{expression} \]

They have a \( Z \) in the first column. The rest is format free. The arguments are considered as dummy arguments of the expression. They must not be followed by a + sign. If the first variable is a number, we are dealing with indexed Z expressions. The number must be in the range 0,127. When no argument or index is present, brackets are absent as well. A section can have several Z expressions, but only one group of substitutions. These substitutions are applied to all Z expressions consecutively.

There are two different kinds of evaluation of an expression (this is true for Z expressions as well as for right-hand sides of substitutions):

i) Working out brackets, replacing special functions and applying built-in operators. This is done immediately and automatically.

ii) All other actions -- substitutions or commands -- are done only at the moment specified by the programmer.
If the Z expression under consideration is the result of a previous run, we are sure that evaluation (i) has no longer to be performed. A faster read-in procedure can then be used. In that case we replace the Z of the first column by an R (for REPEAT). While reading in, the input is converted to the internal notation and written to disk. There can be several R expressions in one section. No substitutions must be given. After the R expressions comes a * YEP card which may then be followed by substitutions and commands.

Z and R expressions cannot be mixed in the same section. The reason is that Z expressions stay in memory while R expressions go directly to disk. This implies as well that there is a limit on the length of the Z expressions, while the R expressions can be (almost) infinitely long.

Example:

\[
\begin{align*}
Z & Z1 = (A+B)\times 2 \\
Z & \text{EXP}(1, U, V) = U \times V \\
Z & \text{EXP}(2, U, V, W) = U \times V + W
\end{align*}
\]

After reading the input, SCHOONSCHIP prints a list called EXPR. which contains the names of all Z expressions, calculated at that moment. In other words, the Z expressions which will be calculated in that section itself are not yet in the list, but those transferred from the previous section are present. In that list, the indexed Z expressions are followed by a number, indicating the value of their index.

### 3.2 Definitions of special functions. X and D expressions

Special functions are replaced by their corresponding expressions whenever they occur. They operate as if they were built-in functions. They are defined in the following format:

\[
X \text{SPFU}(\text{arg1}, \text{arg2}, \ldots) = \text{expression}
\]

and are called X expressions. Their arguments are considered as dummies, but must not have a + sign behind them. When no argument is present, brackets are absent as well. Moreover, there is the indexed X expression (or DATA expression):

\[
D \text{SPFU}(\text{IND}, \text{arg1}, \text{arg2}, \ldots) = \text{term1}, \text{term2}, \ldots \text{termN}.
\]

Term1, term2, ... must be purely multiplicative quantities, e.g. A or A*(B+C) or (B+C). However, A+B is illegal. IND is a dummy index. N \leq 127.

In this way, \text{SPFU}(1, \ldots) will correspond with term1

\[
\begin{align*}
\text{SPFU}(2, \ldots) & \text{ with term2, } \\
\text{SPFU}(N, \ldots) & \text{ with termN.}
\end{align*}
\]

After this definition, \text{SPFU}(\text{EXP}, \ldots) can be used, where \text{EXP} is an integer or an expression that evaluates to an integer in the range \(1, N\).
The definition of an X or D expression can contain another X or D expression. In general, the X or D expression has to be defined before it is allowed to occur on the right-hand side of an expression or substitution.

**Example:**

D ARRAY (N,A,B) = (A+B),(A+B)**2,(A+B)**3  
X FU(N) = DT(N-2)*FU(N-1)/(N-1)  
X UVW = U + V + W + ARRAY(2,U,V)*FU(3)

After reading the input the list EXPR. is printed by SCHOONSCHIP, containing the names of all X and D expressions. They are followed by a number, indicating how many substitution levels are required for their computation.

### 3.3 Declarations of the type of the variables

#### 3.3.1 Defining name lists

It is necessary to give lists that declare the type of all variables which will occur in the formulae. They start in the first column with one of the words: SYMBOL, INDEX, VECTOR, FUNCTION (only the first letter is relevant). Then come the names, separated by commas.

**Example:**

FUNC F1, F2, F3  
VCT P, Q  
I NN, NU, MU

At the same time, the properties of variables with respect to complex conjugation can be specified on these lists. Only the first letter is relevant.

- **IMAGINARY**
- **COMPLEX**
- **UNDEFINED**

Their meaning will be explained later when the operation CONJG is described.

On the I card, the dimension (≤14) of the indices can be indicated (in connection with the rule \( \delta_{\mu\nu} = r = \text{dimension of the space} \)).

**Example:**

INDICES MU = 2, NU = 6

When no dimension is specified, the value 4 is assumed. Moreover, twice occurring indices that are supposed to be summed over should be indicated in the SUM list. (SUM cannot be abbreviated). The SUM card has to be given after the I card. This tells SCHOONSCHIP that the names of those indices are in fact irrelevant, and that they are allowed to be renamed. In this way it is possible to recognize that
E(MU)*F(MU) and E(NU)*F(NU)

(with MU, NU summation indices and E, F functions) are equal.

If a variable is encountered, which is not specified in any list, SCHOONSCHIP applies the following rules to determine its type:

i) Every variable beginning with I, J, K, L, M, N is an index, unless followed by arguments.

ii) Every variable followed by one argument is assumed to be a vector, unless its name consists of three or more characters. Then it is taken to be a function.

iii) Variables followed by several arguments are functions.

iv) Variables of the form V1DV2, where V1 and V2 consist of one or two characters and represent vectors or are undefined, are taken to be dot products. V1 and V2 are taken as vectors.

v) All other variables are algebraic symbols.

3.3.2 Names of COMMON expressions

The validity of these lists extends until a *BEGIN card (see further) which will clear all name lists. When the results of a calculation are written on tape for later use, their associated name lists are automatically stored together with it. Such results are COMMON expressions. In a later run, these lists are read from tape by means of the statement:

NAMES RESULT1, RESULT2, ...

If the name lists of an indexed expression have to be read in, the NAMES card has to specify each value of the index separately, e.g.

NAMES FIA(3), FIA(7)

3.3.3 Printed name lists

After reading all input, SCHOONSCHIP prints out the resulting name lists, which include the built-in quantities. If in one way or the other a conflict arises, i.e. the same variable appears in two lists, then a list CONFUSED will be printed as well, containing all the ambiguous quantities. This is not treated as a fatal error, but it could lead to wrong results.

In this context, there exists the statement

OLDNEW A1 = B1, A2 = B2, ...

which replaces -- at the moment the statement is read -- the old name A1 by the new name B1, etc. The renaming of built-in quantities remains valid until the end of the run.
3.4 Execution instructions

Input is read and translated into internal notation until a card beginning with a * is present. Such a card starts the execution and is the end of a section. Those lists, expressions, etc., which are transferred from one section to the next are determined by the keyword on the * card. We define the following kinds of expressions:

A COMMON expression is an expression which always resides on disk and has its name lists stored together with it.

A KEEP expression is an expression which resides in core if it is not too big, or otherwise on disk. It has no name lists stored together with it.

The programmer has to specify which of his expressions he wants to be COMMON or KEEP.

The action of an execution instruction is given as follows:

* FIX: can only be preceded by name lists, X and D expressions and COMMON cards. These statements are then assumed to be valid throughout that entire run.

* BEGIN: separates two completely different calculations. Only COMMON expressions and "FIX"ed statements remain valid after it. New Z expressions have to be given.

* NEXT: separates two related calculations. Name lists and KEEP expressions remain valid after it. X and D expressions lose their validity. New Z expressions have to be given.

* YEP: subdivides a calculation. The calculation continues with the same Z expressions, the only difference being that a new group of substitutions is given. X and D expressions remain valid. A * YEP can be used when more than 40 substitution levels are required, or when the addition of intermediate results is wanted.

N.B. A * YEP writes the intermediate results on disk and reads them in again, term by term, for further processing.

* END: End of the last part of the calculation. After this execution, no more reading of the input will occur. This defines the end of a run,

* ANYTHING: Any word on a * card, different from NEXT, YEP, END, FIX is interpreted as BEGIN.
3.5 Instructions for the format of the output

The result of a calculation is necessarily represented in the following format:

\[ \text{EXP} = T_1 \times T_2 \times \ldots \times (T_{11} + T_{12} + \ldots) \]
\[ + U_1 \times U_2 \times \ldots \times (U_{11} + U_{12} + \ldots) \]
\[ + \ldots \]

Only one level of bracketing is possible.

3.5.1 Bracket list

All functions and vectors are automatically taken out of brackets. If other quantities are supposed to be taken out of brackets as well, they have to be given in the B-list (Bracket list). There can be at most 25 quantities in the B-list.

Example:

B A3, PDQ, Q(4), ...

(with A3 an algebraic symbol and P, Q vectors).

The validity of the B-list extends over a * YEP, but is cancelled by any other * card.

The appropriate use of the B-card can often speed up the calculation: the scan through the output store, which is necessary for each incoming term, is done by comparing first the factors out of brackets. If those match, each term inside the brackets is inspected. This search is done the fastest when the number of terms inside each pair of brackets is nearly the same as the number of bracket pairs. The order of factors within a term is based on the order of the different quantities in their name lists. The ordering of the terms is at the same time related to their exponents.

3.5.2 Format of numerical coefficients

Terms can be preceded by numerical coefficients. Internally, these coefficients are floating point numbers with a mantissa of 90 bits. Their printing format is:

- integers less than \(2^{16}\) are printed as integers,
- otherwise they are printed as floating point numbers (1 digit before the decimal point and 5 behind).

This format can be changed by using the N card in one of the following forms (valid until a * card):

N 20: Floating point numbers will have 20 digits behind the decimal point (maximum 25).

N R: An attempt will be made to convert each floating point number to the quotient of two integers (within reasonable limits).
N 20,R: If the R attempt is not successful, the printing will be according to N 20.

3.5.3 Print instructions

PRINT STATISTICS) (10 characters relevant)
PRINT NSTATISTICS)
PRINT LIST
PRINT NLIST(T)

Only the first 10 characters -- which include 1 blank -- are relevant. These instructions set ON or OFF the flag for printing of the statistics (running time, number of terms in output, etc.) or of the name lists (vectors, functions, quantities out of brackets, etc.). The list and statistics flags are initially set to ON and are never reset by SCHOONSCHIP itself.

PRINT OUTPUT)
PRINT NOUTPUT)

There exists an over-all output flag and an output flag for each Z expression separately, set ON and OFF by the instructions:

PRINT Z1, Z2, ...
NPRINT Z1, Z2, ...

An expression is printed when both flags (its individual flag and the over-all flag) are ON. At a * BEGIN, * NEXT, * END, the over-all flag is ON by default; at a * YEP it is OFF. The individual flags are always ON by default.

PRINT CINPUT)
PRINT COUTPUT)
PRINT ERROR(R)

These instructions will cause a dump of all relevant quantities (in the internal representation), respectively:

- after analysing the input
- at the end of the calculation
- when an error occurs.

PRINT BRACKETS) (10 characters relevant)

This instruction prints the analysis of the input, while breaking down complex expressions into their elementary subexpressions. The output instructions do not have an immediate effect when read in. They only set flags to signal what printing action has to be taken after execution of the section. Consequently, it does not matter at what place within the section those output instructions are given.
3.6 Instructions to store results

Results can be stored on disk or tape. It depends on the CDC-SCOPE-control cards whether a specific "tape" is a disk file or a magnetic tape.

3.6.1 Results in card image

STORE Z1, Z2, ...

(format free)

The specified expressions are written on TAPE3 in card image format. They can be used as input for a later SCHOONSCHIP program. They are preceded by their name lists and by an R-card.

3.6.2 Results in internal notation

KEEP Z1, Z2, ...

(format free)

The specified expressions -- in internal notation -- are retained in core (or, if too long, on tape) for later use in the same run. The part of the calculation where KEEP files are created has to be terminated with a * NEXT.

COMMON Z1, Z2(0), ...

(format free)

The specified expressions will be written on tape -- in internal notation -- whenever they are created or redefined. In this way they can easily be kept over a * BEGIN. Z2(0) indicates that Z2 will be an indexed file. The COMMON card has to be given together with the declarations of type in the first section.

WRITE COMM(ON)

(10 characters relevant)

This instruction has to be the last step of the calculation. In other words, the program ends with

* BEGIN
WRITE COMMON
* END

All expressions which were COMMON at that moment are now written on TAPE5 -- in internal notation -- and can then be catalogued to be used again in a later SCHOONSCHIP run. Expressions that are COMMON, but should not be written on TAPE5 must be deleted in advance:

DELETE Z1, Z2, ...

(format free)

If a SCHOONSCHIP error occurred during the run, the statement WRITE COMMON will lead to a SCOPE error: ADDRESS OUT OF RANGE. In this way, no execution will be done of the possibly following instructions to purge or catalogue permanent files.

Analogously, there exists an instruction to read in from TAPE5 the COMMON expressions of a previous run: this instruction has to be the first instruction of the new run.
3.6.3 Results on punched cards

Results can be punched on cards, in order to be used later in a FORTRAN program:

```
PUNCH Z1, Z2, ...

The produced cards have a continuation character in the 6th column, and are numbered in columns 73-80.

To produce cards, suitable for SCHOONSCHIP input -- i.e. R expressions -- the cards have first to be written on TAPE3. Then the instructions

TAPE REWIN(D)
TAPE PUNCH
```

punches out the content of TAPE3.

Example:

```
Z EXP = ...
STORE EXP
* BEGIN
TAPE REWIN
TAPE PUNCH
TAPE PUNCH
* END
```

3.7 Substitutions

3.7.1 Internal mechanism

We will now describe the substitution facility. Its general philosophy was already described in Section 1. There are 39 levels at which substitutions can be made. They are characterized by the letter-number combinations L1, L2, ..., L39, or by the keywords

```
ID = (New) IDentity (level) or next free substitution level
AL = ALso or same substitution level.
```

Roughly speaking, the use of ID and AL is an automatic assignment of the levels, while the use of L1, L2, ... implies that the programmer himself has to figure
out how many levels each of his substitutions takes. Still another way to specify levels is to write \( +N \) or \( -N \) (\( N \) = number). The assigned level then equals next free level \( \pm N \). In any case, SCHOONSCHIP prints beside each substitution the \( L.. \) level assigned to it. At a given level, every term of a (partially) evaluated expression is inspected. All substitutions associated with that level are applied to the term. When all terms are treated that way, the level is increased by one and all terms are again inspected for new substitutions belonging to the new level, etc. Working out each pair of brackets is essentially an additional substitution and requires therefore every time an additional level.

Example:

\[
Z\,\text{EXP} = A + B*(C*(F+H)+K)
\]

is analysed as

\[
\begin{align*}
L_1 & \quad \$1 = C*\$2 + K \\
L_2 & \quad \$2 = F + H
\end{align*}
\]

where the dollars are internally created variables. Moreover, a further substitution for \( H \) should be given at level \( L_3 \) or higher, as \( H \) appears only at level \( L_2 \). This counting of levels is done automatically when using the ID feature. In this example:

\[
L_3, \ H = A + B \quad \text{and}
\]

\[
\text{ID, } H = A + B \quad \text{are equivalent.}
\]

It is faster to do several substitutions at the same level rather than applying them consecutively.

The format of a substitution is:

\[
\begin{align*}
\{ \text{ID} \} \\
\{ \text{AL} \} \\
\{ L. . \} \\
\pm N
\end{align*}
\]

3.7.2 Number of levels

"Number of levels" indicates on how many subsequent levels the substitution has to be repeated. If absent, it is taken to be one.

3.7.3 Keywords

"Keywords" are: MULTI, ONCE, FUNCT, DOTPR, AINBE, ADISO.

i) MULTI and ONCE

They apply to substitutions of the type \( A = \ldots \) or \( A**N = \ldots \) with \( A \) an algebraic symbol, an index, a dot product or a vector component. Their action is summarized in the table:
In the latter three substitutions, M and N can be negative integers.

ii) FUNC and DOTPR

When substitutions of the type

\[ A = \ldots \text{ with } A = \text{ algebraic symbol} \]
\[ J = \ldots \text{ with } J = \text{ index} \]
\[ PDQ = \ldots \text{ with } P, Q = \text{ vectors} \]
\[ P(4) = \ldots \text{ with } DU = \text{ dummy} \]

occur, they are not applied to function arguments nor inside dot products. The keyword FUNC makes the substitution applied only to function arguments. The keyword DOTPR ensures that the substitution is applied only to the vectors of dot products and specific vector components with positive exponents (otherwise this could lead to an expression with a negative exponent).

It is impossible to specify two keywords in the same substitution. In such a case the substitution has to be given twice.

Example:

To replace \( P(MU) \) everywhere in a general expression (possibly involving a dot product PDP), one has to give three substitutions:

\[ \text{ID, } P(MU+) = \ldots \] (P is vector, MU is index)
\[ \text{ID, FUNCT, } P(MU+) = \ldots \]
\[ \text{ID, 2, DOTPR, } P(MU+) = \ldots \text{ The } 2, \text{ is necessary to make sure that both vectors in PDP are replaced.} \]

Something like: ID, FUNCT, DOTPR, P(MU+) = ... is illegal.

iii) ADISO and AINBE

Functions are considered to be non-commutative. Substitutions of the type

\[ F1(A1,B1)*F2(A2,B2)* \ldots *F(A,B) = \ldots \]

mean that a matching of this LHS with each incoming term is attempted (scan from left to right). During this scan, the non-commutativity can be partly ignored by specifying:
AINBE (=Allow factors INBEtween) or
ADISO (=Allow DISOrder of the factors).

Example: on the term F1(A)*F2(A)*F3(A)

ID, F1(A)*F2(A) = ... will match ? : yes
ID, F1(A)*F3(A) = ... no
ID, AINBE, F1(A)*F3(A) = ... yes
ID, F3(A)*F2(A) = ... no
ID, ADISO, F3(A)*F2(A) = ... yes.

3.7.4 Left-hand side

The simplest LHS were already mentioned in connection with the keywords. Moreover we have the types (F, F1, F2 are functions):

F(...) = substitute all F for which arguments match
F+(...) = substitute all functions for which arguments match.

Finally more complicated structures can be used in order to do pattern matching.

Example:

ID, A**N+*F1(N+,M+)*F2(N+,K)= ...

Restrictions:

X**N with X an algebraic symbol or
P*(N) with P a vector, are not allowed.

The LHS must be purely multiplicative. If the pattern matches, the substitution will be performed once. It is advisable to test a complicated pattern in a sample program before using it in the actual calculation. If that pattern is not built in, no substitution at all will be made. No error message will be issued either.

3.7.5 Right-hand side

Any legal expression can be on the right-hand side of a substitution.

3.7.6 Examples

i) To illustrate the use of dummies

Z TERM = F(A,B)
ID, F(A,B) = A*B

leads to a replacement: TERM = A*B.

Z TERM = F(A,B)
ID, F(X,Y) = X*Y

does not lead to a replacement: TERM = F(A,B).
i) To illustrate substitutions on algebraic symbols

\[
Z \text{ TERM } = F((A+B),(A-B))
\]

\[
\text{ID, } F(X,Y) = XY
\]

leads to a replacement: \( \text{TERM } = A**2 - B**2 \).

ii) To illustrate substitutions on functions

\[
Z \text{ FAC } = F1(A)*F2(B)*F1(AA)*F2(BB)
\]

\[
\text{ID, } F1(A) = A
\]

leads to \( \text{FAC } = A*AA*F2(B)*F2(BB) \),

\[
\text{ID, } F1(A)*F2(AA) = A*AA
\]

leads to \( \text{FAC } = A*B*F1(AA)*F2(BB) \),

\[
\text{ID, } F2 + (A) = A
\]

leads to \( \text{FAC } = A*AA*BB \).

iii) To illustrate substitutions on algebraic symbols

\[
Z \text{ ALG } = A**4*B**2*C**3*F(A)
\]

\[
\text{ID, } A = AA
\]

leads to \( \text{ALG } = AA**4*B**2*C**3*F(A) \),

\[
\text{ID, } A**2*B**2 = 2B2
\]

does not lead to any change,

\[
\text{ID, } C = CC
\]

does not lead to any change,

\[
\text{ID, ONCE, } A**3 = A3
\]

leads to \( \text{ALG } = A*A3*B**2*C**3*F(A) \).

4. BUILT-IN COMMANDS, FUNCTIONS AND OPERATORS

4.1 Commands

A number of useful commands are built in. As with substitutions, the levels at which the command has to be applied must be specified.

4.1.1 Commands for spinor algebra

i) \( \text{ID, TRICK, I1, I2, ..., TRACE, J1, J2, ...} \)

Reduce products of \( \gamma \)'s of loop \( I1, I2, ... \). Reduce products of \( \varepsilon \)'s.
Take trace of \( \gamma \)'s of loop \( J1, J2, ... \).
Use Chisholm identities and Kahane algorithm as much as possible within the allocated levels (by default 5).

ii) ID, SPIN1, J1, J2, ...

Perform spin summation over loops J1, J2, ... It should be remembered that while doing spin summation, loop indices might be changed. For instance the combination UG(J1,M,P)*UBG(J2,M,P) leads to a change of J2 into J1 everywhere. Thus J2 needs not to be mentioned in the list. However, no harm results from J2 being in the list.

4.1.2 Commands for function arguments

i) ID, EVENX, AF, 1, 2, 4, BF, 2

Recognize AF as a function even in arguments 1, 2 and 4, and BF even in argument 2. Thus remove possible minus signs of these arguments.

Example:
AF(A1,-A2,-A3,P1,B1)
goes over into AF(A1,A2,-A3,P1,B1).

ii) ID, ODDXX, AF, 1, 2, 4, BF, 2

Same as above, only for every minus sign removed the function AF gets a minus sign.

Example:
AF(A1,-A2,-A3,P1,B1)
goes over into -AF(A1,A2,-A3,P1,B1).

iii) ID, REPLA, A1, A2, AF, 1, 2, 4, BF, 2

If argument 1, 2 or 4 of AF or 2 of BF is A1, replace it by A2.

iv) ID, ASYMX, AF, 1, 2, 3, BF, 2, 3

Recognize that AF is antisymmetric in arguments 1, 2 and 3. Set them in a certain (internally well-defined) order, while changing sign if necessary. Same for BF.

Example:
AF(A1,B1,C1,D1) goes over into -AF(C1,B1,A1,D1)
If two arguments are equal the term is made zero.

v) ID, SYMXX, AF, 1, 2, 3, BF, 2, 3

Same as ASYMX, only no sign change.
4.1.3 Commands for matrix manipulation

Essentially the idea is to regroup a product of matrices, like $A_{ij} B_{jk} C_{ki}$ such that equal indices follow each other. Thus we want the result to be $A_{ij} C_{jk} B_{ki}$. For such purposes there is the command

```
ID, ORDER, function name, nr, index1, index2, ...
```
or

```
ID, ORDER, index1, index2, ...
```

which specify (the function and) the index with which the ordering starts.

**Example:** On the term $A(J1,J2)*B(J1,J3,J2)*C(J3,J4)$ the action of $ID$, $ORDER$, $J2$ is as follows:

\[
A(J1,J2)\times B(J1,J3,J2)\times C(J3,J4)
\]

so nothing at all is changed.

The index which follows the function argument that we arrived at is taken as the next ordering index. If it was the last argument, we take the preceding one.

```
ID, ORDER, B, 2
```

means start with second argument of $B$.

\[
A(J1,J2)\times B(J1,J3,J2)\times C(J3,J4)
\]

and leads to

\[
B(J1,J3,J2)\times A(J1,J2)\times C(J3,J4).
\]

Unused factors are left in their original sequence behind the ordered chain. When there is no more continuation possible or when the functions form a closed loop, ordering stops, unless there are still indices specified on the ORDER card. In such a case, the ordering of the unused factor starts with the next index, etc.

4.1.4 Commands for manipulation of orthogonal vectors

i) ID, ORTHG, Q1, P1, P2, P3

This command sets the dot products $Q1DP1$, $Q1DP2$ and $Q1DP3$ to zero.

ii) ID, ORTHN, P1, P2, P3

This command sets $P1DP2$, $P1DP3$ and $P2DP3$ to zero while $P1DP1$, $P2DP2$ and $P3DP3$ are set to one.
4.1.5 **Commands for test on numerical values**

A test on the numerical value of all coefficients during the calculation is performed when using the commands

\[
\begin{align*}
\text{IFSM} & \quad \text{IFSMA (1er)} \\
\text{ID} & \quad \text{IFGRE (ater)}, \ \text{floating point No., algebraic symbol.}
\end{align*}
\]

Every term for which the test:

- coefficient \( \geq \) floating point No.

is successful, gets the specified symbol appended to it. The test is made on each incoming term separately, i.e. before addition of terms is performed.

4.1.6 **Commands to FREEZE a COMMON expression**

It is sometimes helpful to split an expression into invariant pieces and into pieces which one still wants to change. This operation is called "Freezing".

Assume FIA is a COMMON, unindexed expression. After evaluation, it has necessarily the form

\[\text{TT1*}(T1+T2+T3+...) + \text{TT2*}(U1+U2+U3+...) + \text{TT3*}(V1+V2+V3+...) + ...\]

The statements

* BEGIN
FREEZE FIA
* BEGIN

replace FIA by \( \text{TT1*DF(FIA,1) + TT2*DF(FIA,2) + TT3*DF(FIA,3) + ...} \)

with

\[
\begin{align*}
\text{DF(FIA,1)} & = T1 + T2 + T3 + ... \\
\text{DF(FIA,2)} & = U1 + U2 + U3 + ... \\
\text{DF(FIA,3)} & = V1 + V2 + V3 + ...
\end{align*}
\]

DF is a special function, the purpose of which is to make the frozen subexpressions addressable. If the initial COMMON expression had arguments, then DF has those arguments as well:

\[\text{DF(FIA, 1, arg1, arg2, ...).}\]

The command

ID, EXPAND, FIA

replaces the DF functions by their original expressions. The section where this command is given, has to contain the card: NAMES FIA.
In the name list EXPR., the name of the frozen expression will appear, followed by a number indicating how many frozen subexpressions it has. This number can be as high as 4096. This means that, for example, DF(FIA,3000) is legal. In other words, at this place -- but only at this place -- an integer > 128 is accepted without being truncated.

The "Freezing" procedure is useful when one knows in advance that the next set of substitutions will only affect TT1, TT2, ... and not T1, T2, ..., U1, ... .

4.1.7 Various commands

i) ID, NUMER, algebraic symbol
    vector component
    index
    dot product
    { floating point No., ...

This is an efficient way to give a numerical value to a quantity.

Example:
ID, NUMER, A, 1, P(2), 7, B, 3.2E-3

ii) ID, RATIO, XA, XB, BA

This command performs rationalization or partial fraction decomposition. With this we mean a replacement of the type:

\[
\frac{1}{(x + a)^m(x + b)^n} = \sum_{k=1}^{m} \frac{A_k}{(x + a)^k} + \sum_{k=1}^{n} \frac{B_k}{(x + b)^k}
\]

or

\[
\frac{(x + b)^n}{(x + a)^m} = \sum_{k} \frac{C_k}{(x + a)^k} + \text{possibly a polynomial in } (x + a) .
\]

A_k, B_k, and C_k depend only on (b - a).

We introduce the notation:
XA = x + a, XB = x + b, BA = b - a .

Then RATIO acts on XA**M*XB**N.

Example:
XA**(-2)*XB**(-2) leads to XA**(-2)*BA**(-2) - 2.*XA**(-1)*BA**(-3)
    + XS**(-2)*BA**(-2) + 2.*XB**(-1)*BA**(-3)

XA**(-3)*XB**2 leads to XA**(-3)*BA**2 + 2.*XA**(-2)*BA + XA**(-1) .
iii) \[
\begin{align*}
\text{ID, COUNT,} & \quad \{ \text{dot product} \} \quad \begin{cases} 
\text{number} \\
\text{vector component} \\
\text{algebraic symbol}
\end{cases} \\
, \quad \{ \text{quantity, No.} \} \quad \begin{cases} 
\text{index} \\
\text{function} \\
X \text{ expression}
\end{cases}
\end{align*}
\]

The execution of this command goes in two steps:

a) The list: quantity, No., ... assigns a weight to each quantity. The weight has to be between $-2^7$ and $2^7$. The weight of a product is the sum of the weights of all factors. The weight of function arguments is not counted. The weight of a vector component or dot product cannot be given on the COUNT card, but is computed on the basis of the weight of the individual vectors. In this way, the weight of each incoming term is computed.

b) The action taken depends on the variable specified behind the keyword COUNT

1. number: all terms with weight < number are set to zero
2. vector component
   - algebraic symbol
   - dot product
   - index
   a factor "variable ** weight" is appended to each term
3. function
   - X expression
   a factor "variable (weight)" is appended to each term

**Example:** On the term PDP (P is a vector, F is a function, A is an algebraic symbol)

ID, COUNT, i5, r, c leads to u,
ID, COUNT, F, P, 5 leads to PDP*F(10),
ID, COUNT, A, P, 5 leads to PDP*A**10.

iv) ID, COMPO, funct1, funct2, ...

This command COMPOses names for algebraic symbols and rearranges function arguments at the same time.

a) \(F(=B,=C,=A)\) leads to \(F(BCA)\) (specified characters)
\(F(=A,3,12)\) leads to \(F(A312)\) (numbers must be positive)
\(F(G2,MM,B)\) leads to \(F(G2MMB)\) (names are joined together, max. 5 characters)

by means of the command: ID, COMPO, F. F must be a function. The quantity "=L" is everywhere accepted as a legal function argument.
b) When F has additional arguments, separated with *, then the constructed
name has its characters ordered. The additional arguments get permuted
in the same way as the characters of the name.

\[ F(=B,=C,=A,*,...,B1,*,C1,C2,C3,*,A1,A2) \] leads to \[ F(ABC,A1,A2,B1,C1,C2,C3). \]

Obviously, the number of arguments before the first star must be equal
to the number of stars.

The rule for the ordering is:

- \[ F(=B,=C,=A,*,...,...) \] becomes \[ F(ABC,...) \] alphabetical order.
- \[ F(25,=A,7,*,...,...) \] becomes \[ F(A72,...) \] letters before numbers, numbers
  increasing order. Only the first character is taken.
- \[ F(G2,MM,B,*,...,...) \] becomes \[ F(MGB,...) \] if \( G2 \) is a function, \( MM \) an index,
  and \( B \) an algebraic symbol. The names are ordered according to the
  name lists. The order is: index, vector, function, algebraic symbol,
  vector component, dot product. Then of each the first letter is taken.

The quantity "*" is everywhere accepted as a legal function argument.
All the created names which did not exist in a name list are inserted
in the algebraic symbol list.

4.2 The built-in functions

A number of special functions are built in. Names, number of arguments ex-
pected, and meaning are:

<table>
<thead>
<tr>
<th>Name</th>
<th>No. of arg.</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>2</td>
<td>D(M,N)</td>
<td>delta function for spinor algebra</td>
</tr>
<tr>
<td>EPF</td>
<td>any</td>
<td>EPF(M,N,L)</td>
<td>totally antisymmetric tensor</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>G(J,MU)</td>
<td>( \gamma^I ) of loop J</td>
</tr>
<tr>
<td>GI</td>
<td>1</td>
<td>GI(J)</td>
<td>1 of loop J</td>
</tr>
<tr>
<td>G5</td>
<td>1</td>
<td>G5(J)</td>
<td>( \gamma^5 ) of loop J</td>
</tr>
<tr>
<td>G6</td>
<td>1</td>
<td>G6(J)</td>
<td>( 1 + \gamma^5 ) of loop J</td>
</tr>
<tr>
<td>G7</td>
<td>1</td>
<td>G7(J)</td>
<td>( 1 - \gamma^5 ) of loop J</td>
</tr>
<tr>
<td>UG</td>
<td>3 or 4</td>
<td>UG(J,A,P)</td>
<td>spinor ( u(J,A,P) ) of loop J</td>
</tr>
<tr>
<td>UBG</td>
<td>3 or 4</td>
<td>UBG(J,A,P)</td>
<td>spinor ( \bar{u}(J,A,P) ) of loop J</td>
</tr>
</tbody>
</table>

These functions are extensively discussed in Section 1.2. In UG and UBG the
symbols \( A \) and \( P \) stand for mass and four-momentum. For antiparticles one should
use \(-A\) instead of \( A, P \) remaining as it stands. These special functions obey the
rules of vector and spinor algebra given before. A spin 3/2 spinor is represented
by UG(J,MU,A,P). We emphasize that \( A \), representing a mass, must be an algebraic
symbol, P a vector, and J an index. MU may be index or vector. An expression UG(J,P,A) where P and A are vector and algebraic quantity, respectively, leads to erratic behaviour of the program.

<table>
<thead>
<tr>
<th>Name</th>
<th>No. of arg.</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD</td>
<td>2</td>
<td>DD(J,K)</td>
<td>Replace every J by K. Here J and K are not numerical. e.g. F(J)*DD(J,-K) leads to F(-K). Note the behaviour of the minus sign. It is different from the D function.</td>
</tr>
<tr>
<td>DD</td>
<td>2</td>
<td>DD(J,K)</td>
<td>= J/K. Here J and K are numerical.</td>
</tr>
<tr>
<td>DD</td>
<td>1</td>
<td>DD(J)</td>
<td>= 1/J. J must be numerical.</td>
</tr>
<tr>
<td>DB</td>
<td>2</td>
<td>DB(N,M)</td>
<td>Binomial function. N and M are numerical. DB = ( \frac{n!}{m!(n-m)!} ).</td>
</tr>
<tr>
<td>DT</td>
<td>any</td>
<td>DT(N1,N2,N3)</td>
<td>Product of theta functions. DT = 1 if N1 ≥ 0 and N2 ≥ 0 and N3 ≥ 0. DT = 0 in all other cases. N.B. -0 is considered equal to +0.</td>
</tr>
<tr>
<td>DS</td>
<td>4 or 5</td>
<td>DS(J,K,N,(F(J)),(H(J)))</td>
<td>Summation function. DS = ( \sum_{j=k}^{m} f(j) \cdot g(j) ) where g(k) = 1; g(m) = g(m-1) \cdot h(m); h(j) can only be a numerical expression. The last argument of the DS function may be absent. This implies h(j) = 1. When K &gt; N, DS = 0 is taken. J is a dummy which must not be declared in any name list. I should not be used for any other purpose in that section.</td>
</tr>
<tr>
<td>DX</td>
<td>any</td>
<td>DX(4,N,(M))</td>
<td>Link to user provided FORTRAN subroutine EXTRA (I,A,R). R is the double precision floating point result of the FORTRAN calculation and is transferred as value for DX. I is an integer array (dimension 10) which contains those arguments of DX which are not in brackets (consequently integers between -128 and +128). A is a double precision array (arbitrary dimension) which contains the other arguments of DX (consequently floating point numbers).</td>
</tr>
</tbody>
</table>
4.3 The built-in operators

Two operators are available:

CONJG: which takes the complex conjugation,

INTEG: which does the numerical evaluation of an expression and truncates the result to an integer.

4.3.1 Complex conjugation

In order to make complex conjugation possible the reality properties of the quantities involved must be given. The reality properties of algebraic symbols and functions are given in the S- and F-lists, respectively; all other quantities must be given. The reality properties of algebraic symbols

For algebraic symbols the notation in the S-list:

..., A = I, ... implies A is imaginary.

..., A = C, ... implies A is complex. The program generates a new symbol, by appending a G, which denotes the complex conjugate of A. Here this would be AG.

..., A, ... implies A is real.
In the F-list:

..., Fl = I,... implies Fl is imaginary;
..., Fl = C,... implies Fl is complex, with FlG as new symbol denoting
the complex conjugate of Fl;
..., Fl = U,... implies that the properties of Fl under complex conjuga-
tion are undefined, and complex conjugation for such a
function should be postponed till in some substitution
some expression is substituted for Fl. Thus, during
execution time the symbol CONJG remains attached to Fl
till some substitution is made.
..., Fl,... implies F is real.

Example:

SYMBOLS A = I, B1, B2 = COMPL

Then:

CONJG(A+B1+B2) = -A + B1 + B2G.

With respect to functions CONJG reverses the order of the functions and performs
complex conjugation as above on each function and its arguments separate.

Example:

FUNCTIONS ALF = C, ALS, F1, F2 = IMAG, F3 = U

The S-list is as above. One obtains:

CONJG(ALF(A,B1)*F1(B2)*F2(A,B2,(A+B1))*F3(A)*F3(B1))
= - CONJG(F3(A)*F3(B1))*F2(-A,B2G,CONJG(A+B1))*F1(B2G)*ALFG(-A,B1).

No "U" functions should appear as arguments of non "U" functions.

Moreover, it is understood that G7 = CONJG(G6) and UBG = CONJG(UG).

4.3.2 The INTEG operator

Whenever a floating point number (coefficient of a term) has to be converted
to an integer (e.g. because it appears as a function argument or as an exponent),
the operator INTEG has to be used.

F(A,(2+3),B) leads internally to F(A,$1,B) and $1 = 5.0.

The subsequent substitution ID, F(U*,V*,W*) = U**V + W leads to A**$1 + B and
will produce an error, because an exponent must be a number and never an expression.

On the other hand F(A,INTEG(2+3),B) leads internally to F(A,5,B) and every-
thing will go correctly now.

In most of the cases, SCHUSCHIP generates automatically the INTEG in front
of expressions:
F(A,2+3,B) is converted at read in time to F(A,INTEG(2+3),B). The rule is that every time an expression is seen at a place where its appearance is illegal, the operator INTEG is set in front of it. As a consequence INTEG is used explicitly only very rarely, for example the following case:

\[
\begin{align*}
Z \text{ NUMB} & = 5 \\
\text{KEEP NUMB} \\
* \text{ NEXT} \\
Z \text{ EXP} & = F(\text{NUMB}) \\
\text{ID}, F(N+) & = A**N
\end{align*}
\]

will lead to an error: SCHOONSCHIP does not see any reason to generate \( \text{EXP} = F(\text{INTEG(\text{NUMB}})) \) which is necessary to obtain the result \( \text{EXP} = A**5 \). So either one has to insert the INTEG operator by hand: \( Z \text{ EXP} = F(\text{INTEG(\text{NUMB}})) \) or force it to appear by writing, e.g. \( Z \text{ EXP} = F(\text{NUMB} + 0) \) or \( Z \text{ EXP} = F(1*\text{NUMB}) \).

N.B. INTEG converts -0 to +0.

5. INPUT FOR SCHOONSCHIP

5.1 Format for punched cards

A comment card is a card with a C in the first column. All 80 columns can be used. A card with a blank first column is a continuation of the last preceding statement, which was not a comment. If all preceding cards are comment cards, the continuation card is also treated as a comment. Only 72 columns can be used on cards that are not comment cards.

Most statements are format free. By this we mean:

- A first keyword (or just one letter) has to be punched starting in column one. Otherwise the card is considered as a continuation card.
- A blank or comma ends the keyword.
- In the other cases, blanks are ignored.

Example:

\[
\begin{align*}
\text{ID,5,MULTI,A**N=} & \ldots \text{ can also be punched} \\
\text{ID, 5 MULTI, A * N =} \\
or \\
F \text{F1,F2,F3} & \text{ can also be punched} \\
\text{FUNCTIONS , F1 , F2 , F3}
\end{align*}
\]

The flag that controls the printing of the input is initially ON, and reset after every * card. It can be changed with the instructions

\[
\begin{align*}
\text{PRINT INPU(T)} & \quad \text{(10 characters relevant)} \\
\text{PRINT NINP(UT)},
\end{align*}
\]
EXAMPLE, EXPANSION IN POWER SERIES

Schonheits, Version of January 1, 1973

TIME... .02 SECONDS

C EXAMPLE, EXPANSION IN POWER SERIES UP TO TENTH ORDER IN Y

FUNCTION FS,Y
C SERIES TO BE EXPANDED.
7 EXPANSIONS (3X**2+2)/X(4+Y/3))=>X-FC((2X*Y))/Y**4 + A * B
C THIS EXPRESSION IS NOW EXAMINED FOR SUBSTITUTIONS.
C
FIRST SUBSTITUTION. 2X MEANS REPLACE FOR ANY VALUE OF Z
C WHEREVER IN EXPAN THE FACTOR FC IS SEEN, FC IS REPLACED BY THE
C RIGHT HAND SIDE OF THE SUBSTITUTION.
C
SECOND SUBSTITUTION.
10, FC((2*Z)*4+Z**2+3)/6-Z**5/125-Z**7/240+Z**9/330+Z**11/399160
C WHEREVER IN EXPAN THE FACTOR FC IS SEEN, FC IS REPLACED BY THE
C RIGHT HAND SIDE OF THE SUBSTITUTION.
C
AFTER THESE SUBSTITUTIONS, ALL TERMS BECOME SIMPLIFIED.
C
AS NEXT TO PROCEDURE, USE OF EXPANSION AT X=0.
C
MULTIPLY EXPANSION BY X**10
C
END
C
SYMBOLS
X, Y, A, B, C

FUNCTIONS
4, EXP., G4, D1, D2, G10, G20, G50, G60, G80, G90, G100, G110,
DR. G200, G201, G202, G203

RUNNING TIME (SEC)... 1.24
USED 164 KB

TERMS IN JUMP... 0
GENERATED TERMS... 0
EQUAL TERMS... 0
CANCELLATIONS... 9
RECIPROCAL WRITTEN... 0
MULTIPLICATIONS... 10

EXPANSION

* 5.33333 + 1.58274E1 Y**2 + 2.10974E4 Y**4 + 1.55601E8 Y**6 + 1.39056E14 Y**8 + 8.24569E18 Y**10 +

END OF RUN, TIME... 1.32 SECONDS
APPENDIX 2

EXAMPLE. A CALCULATION FROM HIGH-ENERGY PHYSICS

SCHOOLSCIP, VERSION OF JANUARY 1, 1973

TIME 1.33 SECONDS

C EXAMPLE. A CALCULATION FROM HIGH-ENERGY PHYSICS,

C DECAY OF KAON INTO PION + MUON + NEUTRINO, WITH AXIAL MAGNETIC TERM IN
C LEPTON PART.

C NOTATIONS FOR MASS AND MOMENTUM
C KAON (PLUS) MK
C PION MP
C MUON M
C NEUTRINO MN
C W = POLARIZATION DIRECTION.

FUNCTION DIAMOND DEFINED
SYMBOLS LA=COMPLEX, KSIG=COMPLEX, F1, M, MN, MK, MP.
VECTORS Q, P=K, W.
INDICES MU, NU, MUP, NUP.

L 2 Z KAON = (MU, NU) + CONJG(DIAMOND)
L 5 I0, I1 = MU, NU, IUP
L 6 I0, I1 = MU, NU, IUP
L 11 AL, DOP = MU, NU
L 11 AL, DOP = MU, NU
L 11 AL, DOP = MU, NU
L 11 AL, DOP = MU, NU
L 11 AL, DOP = MU, NU

C LENGTH = IMAGINARY PART KSI
C RIA = REAL PART KSI
C ILA = IMAGINARY PART LA
C RIA = REAL PART LA

SYMBOLS F1, F2, D, E, T, Q, P, K, W.

INDICES M1, M2, M3, M4, M5, M6, M7.

VECTORS Q, P = K, W.

FUNCTIONS DIA, DIA, DIA, DIA, DIA, DIA.

RUNNING TIME (SEC) 1.33
GENERATED TERMS 420
INPUT SPACE 520
EQUAL TERMS 347
OUTPUT SPACE 1150
CANCELLATIONS 125
NR. OF EXPRS. 51
RECORDS WRITTEN 24
ID. REGISTER 50
MULTIPLICATIONS 111,853
FUNCT. REG. 102

SYMBOLS RLA, ILA, KSIG, IKSIG
C RLA = REAL PART OF LA
C ILA = IMAGINARY PART OF LA
C KSIG = REAL PART OF KSIG
C IKSIG = IMAGINARY PART OF KSIG

END

SYMBOLS F1, F2, D, E, T, Q, P, K, W.

INDICES M, N, MUP, MUP, M, N, M, MUP, ILA, KSIG.
VECTORS  U, OP, K, M.

FUNCTIONS  B, PF, G1, G2, G3, G4, G5, G6, G7, UGE, UGH, NDA, NH,
            N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11,
            N12, N13, N14, N15, N16, N17, N18, N19, N20,
            N21, N22, N23, N24, N25, N26, N27, N28, N29,
            N30, N31, N32, N33, N34, N35, N36, N37, N38,
            N39, N40, N41, N42, N43, N44, N45, N46, N47,
            N48, N49, N50, N51, N52, N53, N54, N55, N56,
            N57, N58, N59, N60, N61, N62, N63, N64, N65,
            N66, N67, N68, N69, N70, N71, N72, N73, N74,
            N75, N76, N77, N78, N79, N80, N81, N82, N83,
            N84, N85, N86, N87, N88, N89, N90, N91, N92,
            N93, N94, N95, N96, N97, N98, N99, N100, N101,
            N102, N103, N104, N105, N106, N107, N108, N109,
            N110, N111, N112, N113, N114, N115, N116, N117,
            N118, N119, N120, N121, N122, N123, N124, N125,
            N126, N127, N128, N129, N130, N131, N132, N133,
            N134, N135, N136, N137, N138, N139, N140, N141,
            N142, N143, N144, N145, N146, N147, N148, N149,
            N150, N151, N152, N153, N154, N155, N156, N157,
            N158, N159, N160, N161, N162, N163, N164, N165,
            N166, N167, N168, N169, N170, N171, N172, N173,
            N174, N175, N176, N177, N178, N179, N180, N181,
            N182, N183, N184, N185, N186, N187, N188, N189,
            N190, N191, N192, N193, N194, N195, N196, N197,
            N198, N199, N200, N201, N202, N203, N204, N205,
            N206, N207, N208, N209, N210, N211, N212, N213,
            N214, N215, N216, N217, N218, N219, N220, N221,
            N222, N223, N224, N225, N226, N227, N228, N229,
            N230, N231, N232, N233, N234, N235, N236, N237,
            N238, N239, N240, N241, N242, N243, N244, N245,
            N246, N247, N248, N249, N250, N251, N252, N253,
            N254, N255, N256, N257, N258, N259, N260, N261,
            N262, N263, N264, N265, N266, N267, N268, N269,
            N270, N271, N272, N273, N274, N275, N276, N277,
            N278, N279, N280, N281, N282, N283, N284, N285,
            N286, N287, N288, N289, N290, N291, N292, N293,
            N294, N295, N296, N297, N298, N299, N300, N301,
            N302, N303, N304, N305, N306, N307, N308, N309,
            N310, N311, N312, N313, N314, N315, N316, N317,
            N318, N319, N320, N321, N322, N323, N324, N325,
            N326, N327, N328, N329, N330, N331, N332, N333,
            N334, N335, N336, N337, N338, N339, N340, N341,
            N342, N343, N344, N345, N346, N347, N348, N349,
            N350, N351, N352, N353, N354, N355, N356, N357,
            N358, N359, N360, N361, N362, N363, N364, N365,
            N366, N367, N368, N369, N370, N371, N372, N373,
            N374, N375, N376, N377, N378, N379, N380, N381,
            N382, N383, N384, N385, N386, N387, N388, N389,
            N390, N391, N392, N393, N394, N395, N396, N397,
\* IKS**2 \\
\* ( - 2.1*M**2*HH**2 + 2.1*M**2*HR**2 - 4.1*M**2*QDK - 4.1*M**2*QPD - 2.1*N**4 ) \\

\* EPF(Q,OP,K,W)*RLA+IKSI+F1 \\
\* ( 16,1)*H ) \\

\* EPF(Q,OP,K,W)*ILA+H**2 \\
\* ( 16,1)*H + 32.1*M**(-1)*QDK - 32.1*M**(-1)*QPD ) \\

\* EPF(Q,OP,K,W)*ILA+RKS+*F1 \\
\* ( - 16,1)*H ) \\

\* EPF(Q,OP,K,W)*IKSI+F1 \\
\* ( - 16,1)*H ) *0. \\

END OF RUN,   TIME     8.33 SECONDS
<table>
<thead>
<tr>
<th>Statement</th>
<th>Short description</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressions to be evaluated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z EXP(A,B) = ...</td>
<td>Z expression, To be evaluated</td>
<td>13</td>
</tr>
<tr>
<td>R EXP(A,B) = ...</td>
<td>R(epeated) expression. Input from previous run.</td>
<td>14</td>
</tr>
<tr>
<td>Definition of special functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X F = ...</td>
<td>X expression.</td>
<td>14</td>
</tr>
<tr>
<td>D F(N) = ..., ..., ...,</td>
<td>D(data) expression. Indexed X expression.</td>
<td>14</td>
</tr>
<tr>
<td>Declaration of the type of the variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUNCTION F1,F2</td>
<td>Function name list.</td>
<td>15</td>
</tr>
<tr>
<td>INDEX I1,I2</td>
<td>Index name list.</td>
<td>15</td>
</tr>
<tr>
<td>SYMBOL S1,S2</td>
<td>Symbol name list.</td>
<td>15</td>
</tr>
<tr>
<td>VECTOR V1,V2</td>
<td>Vector name list.</td>
<td>15</td>
</tr>
<tr>
<td>SUM I1,I2</td>
<td>List of summation indices.</td>
<td>15</td>
</tr>
<tr>
<td>... = COMPLEX</td>
<td>Declaration of reality properties of functions, algebraic symbols.</td>
<td>15</td>
</tr>
<tr>
<td>... = IMAGINARY</td>
<td>Declaration of reality properties of functions, algebraic symbols.</td>
<td>15</td>
</tr>
<tr>
<td>... = UNDEFINED</td>
<td>Declaration of reality properties of functions</td>
<td>15</td>
</tr>
<tr>
<td>... = number</td>
<td>Declaration of dimension of an index.</td>
<td>15</td>
</tr>
<tr>
<td>NAMES Z1,Z2</td>
<td>Reads in the name lists of the COMMON expressions Z1,Z2.</td>
<td>16</td>
</tr>
<tr>
<td>OLDNEW A1 = A2</td>
<td>Renames a quantity.</td>
<td>16</td>
</tr>
<tr>
<td>Execution instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* FIX</td>
<td>Makes valid for the entire run.</td>
<td>17</td>
</tr>
<tr>
<td>* BEGIN</td>
<td>Transfer of COMMON files.</td>
<td>17</td>
</tr>
<tr>
<td>* NEXT</td>
<td>Transfer of KEEP files.</td>
<td>17</td>
</tr>
<tr>
<td>* YEP</td>
<td>Generates intermediate results.</td>
<td>17</td>
</tr>
<tr>
<td>* END</td>
<td>End of run.</td>
<td>17</td>
</tr>
<tr>
<td>Statement</td>
<td>Short description</td>
<td>Page No.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Instructions for the format of the output</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B A1,A2</td>
<td>List of quantities out of brackets.</td>
<td>18</td>
</tr>
<tr>
<td>N number</td>
<td>Number of decimals for floating point numbers.</td>
<td>18</td>
</tr>
<tr>
<td>N R</td>
<td>Floating point numbers to be converted to ratio of two integers.</td>
<td>18</td>
</tr>
<tr>
<td>PRINT STATISTICS</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT NSTATISTICS</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT LIST</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT NLIST</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT OUTPUT</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT NOUTPUT</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>PRINT CINPUT</td>
<td>Dump request.</td>
<td>19</td>
</tr>
<tr>
<td>PRINT COUTPUT</td>
<td>Dump request.</td>
<td>19</td>
</tr>
<tr>
<td>PRINT ERROR</td>
<td>Dump request.</td>
<td>19</td>
</tr>
<tr>
<td>PRINT BRACKETS</td>
<td>Analysis of input.</td>
<td>19</td>
</tr>
<tr>
<td>PRINT Z1,Z2 ...</td>
<td>Sets individual print flag ON.</td>
<td>19</td>
</tr>
<tr>
<td>NPRINT Z1,Z2 ...</td>
<td>Sets individual print flag OFF.</td>
<td>19</td>
</tr>
<tr>
<td><strong>Instructions to store results</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE Z1,Z2, ...</td>
<td>Writes on TAPE3 in card image.</td>
<td>20</td>
</tr>
<tr>
<td>KEEP Z1,Z2, ...</td>
<td>Transfers over * NEXT in internal notation.</td>
<td>20</td>
</tr>
<tr>
<td>COMMON Z1,Z2</td>
<td>Transfers over * BEGIN in internal notation.</td>
<td>20</td>
</tr>
<tr>
<td>DELETE Z1,Z2</td>
<td>Deletes COMMON expressions.</td>
<td>20</td>
</tr>
<tr>
<td>WRITE COMMON</td>
<td>Writes COMMON expressions in internal notation on TAPE5 (for later run).</td>
<td>20</td>
</tr>
<tr>
<td>ENTER COMMON</td>
<td>Reads COMMON expressions from TAPE5 (in later run).</td>
<td>21</td>
</tr>
<tr>
<td>PUNCH Z1,Z2</td>
<td>Punches cards for use in FORTRAN.</td>
<td>21</td>
</tr>
<tr>
<td>TAPE PUNCH</td>
<td>Punches content of TAPE3.</td>
<td>21</td>
</tr>
<tr>
<td><strong>Level of substitutions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL</td>
<td>Assigns the same substitution level.</td>
<td>21</td>
</tr>
<tr>
<td>ID</td>
<td>Assigns the next free substitution level.</td>
<td>21</td>
</tr>
<tr>
<td>Statement</td>
<td>Short description</td>
<td>Page No.</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Keywords for substitutions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTI</td>
<td>Applies to A**N=...</td>
<td>22</td>
</tr>
<tr>
<td>ONCE</td>
<td>Applies to A**N=...</td>
<td>22</td>
</tr>
<tr>
<td>FUNCT</td>
<td>Replacement inside functions.</td>
<td>23</td>
</tr>
<tr>
<td>DOTPR</td>
<td>Replacement inside dot products and specific vector components.</td>
<td>23</td>
</tr>
<tr>
<td>ADISO</td>
<td>Treats functions as commuting (Allow disorder).</td>
<td>23</td>
</tr>
<tr>
<td>AINBE</td>
<td>Treats functions as partly commuting (Allow in between).</td>
<td>23</td>
</tr>
<tr>
<td><strong>Commands for spinor algebra</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRICK, TRACE</td>
<td>Reduction of ε's and γ's. Trace of γ's.</td>
<td>25</td>
</tr>
<tr>
<td>SPIN1</td>
<td>Spin summation</td>
<td>26</td>
</tr>
<tr>
<td><strong>Commands for function arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVENX</td>
<td>Even function.</td>
<td>26</td>
</tr>
<tr>
<td>ODDXX</td>
<td>Odd function.</td>
<td>26</td>
</tr>
<tr>
<td>REPLA</td>
<td>Replacement of function arguments.</td>
<td>26</td>
</tr>
<tr>
<td>ASYMX</td>
<td>Asymmetric function.</td>
<td>26</td>
</tr>
<tr>
<td>SYMXX</td>
<td>Symmetric function.</td>
<td>27</td>
</tr>
<tr>
<td><strong>Commands for matrix manipulation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORDER</td>
<td>Order functions.</td>
<td>27</td>
</tr>
<tr>
<td><strong>Commands for orthogonal vectors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORTHG</td>
<td>Orthogonal vectors.</td>
<td>27</td>
</tr>
<tr>
<td>ORTHN</td>
<td>Orthonormal vectors.</td>
<td>27</td>
</tr>
<tr>
<td><strong>Commands to test on numerical values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFEQU</td>
<td>Append symbol if coefficient is equal, greater or smaller than the given number.</td>
<td>28</td>
</tr>
<tr>
<td>IFGRE</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>IPSMA</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td><strong>Commands to freeze a COMMON expression</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREEZE</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>EXPAND</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Statement</td>
<td>Short description</td>
<td>Page No.</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Various commands</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMER</td>
<td>Assigns a numerical value.</td>
<td>29</td>
</tr>
<tr>
<td>RATIO</td>
<td>Partial fraction decomposition.</td>
<td>29</td>
</tr>
<tr>
<td>COUNT</td>
<td>Considers weight of terms.</td>
<td>30</td>
</tr>
<tr>
<td>COMPO</td>
<td>Composes names. Rearranges function arguments.</td>
<td>30</td>
</tr>
<tr>
<td><strong>The built-in functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPF</td>
<td>Epsilon function. Totally antisymmetric tensor.</td>
<td>31</td>
</tr>
<tr>
<td>G</td>
<td>Dirac matrix $\gamma^\mu$.</td>
<td>31</td>
</tr>
<tr>
<td>GI</td>
<td>Unit Dirac matrix.</td>
<td>31</td>
</tr>
<tr>
<td>G5</td>
<td>$\gamma^1 \gamma^2 \gamma^3 \gamma^5$.</td>
<td>31</td>
</tr>
<tr>
<td>G6</td>
<td>$1 + \gamma^5$.</td>
<td>31</td>
</tr>
<tr>
<td>G7</td>
<td>$1 - \gamma^5$.</td>
<td>31</td>
</tr>
<tr>
<td>UG</td>
<td>Spinor for spin $\frac{1}{2}$ or $\frac{3}{2}$ particles.</td>
<td>31</td>
</tr>
<tr>
<td>UBG</td>
<td>Complex conjugate of UG.</td>
<td>31</td>
</tr>
<tr>
<td>D</td>
<td>Delta function of spinor algebra.</td>
<td>31</td>
</tr>
<tr>
<td>DD</td>
<td>Replacement function or numerical division.</td>
<td>32</td>
</tr>
<tr>
<td>DX</td>
<td>Kronecker delta function (for numerical arguments).</td>
<td>33</td>
</tr>
<tr>
<td>DB</td>
<td>Binomial function.</td>
<td>32</td>
</tr>
<tr>
<td>DP</td>
<td>Permutation function.</td>
<td>33</td>
</tr>
<tr>
<td>DT</td>
<td>Product of theta functions.</td>
<td>32</td>
</tr>
<tr>
<td>DX</td>
<td>Function to call external FORTRAN subroutine.</td>
<td>32</td>
</tr>
<tr>
<td>DS</td>
<td>Summation function.</td>
<td>32</td>
</tr>
<tr>
<td>DF</td>
<td>Function to reference frozen subfiles.</td>
<td>33</td>
</tr>
<tr>
<td><strong>The built-in operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONJG</td>
<td>Complex conjugation.</td>
<td>33</td>
</tr>
<tr>
<td>INTEG</td>
<td>Operator for numerical evaluation and conversion to integer.</td>
<td>34</td>
</tr>
<tr>
<td><strong>Input instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Comment card.</td>
<td>35</td>
</tr>
<tr>
<td>PRINT INPUT</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>PRINT NINPUT</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Statement</td>
<td>Short description</td>
<td>Page No.</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>TAPE REWIND</td>
<td>Copy input to TAPE3 until T is read.</td>
<td>36</td>
</tr>
<tr>
<td>TAPE ENDFILE</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>TAPE COPY</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>STORE Z1, Z2</td>
<td>Write name lists, results on TAPE3.</td>
<td>36</td>
</tr>
<tr>
<td>TAPE READ</td>
<td>Read TAPE3 until TAPE END is read.</td>
<td>36</td>
</tr>
<tr>
<td>TAPE FORWARD</td>
<td>Skip }</td>
<td></td>
</tr>
<tr>
<td>TAPE END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAPE NAMES</td>
<td>Read TAPE3 until TAPE START is read.</td>
<td>36</td>
</tr>
<tr>
<td>TAPE NONAMES</td>
<td>Skip }</td>
<td></td>
</tr>
<tr>
<td>TAPE START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAPE PRINT</td>
<td>Print TAPE3 until any tape instruction is read.</td>
<td>36</td>
</tr>
<tr>
<td>TAPE PUNCH</td>
<td>Punch }</td>
<td></td>
</tr>
</tbody>
</table>

**Interactive use**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Short description</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>Starts } interactive use.</td>
<td>37</td>
</tr>
<tr>
<td>PRINTER</td>
<td>Ends }</td>
<td>37</td>
</tr>
</tbody>
</table>
INDEX OF SCHOONSCHIP STATEMENTS

<table>
<thead>
<tr>
<th>INDEX</th>
<th>SHORT DESCRIPTION</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID,AISO</td>
<td>ALLOW DISORDER FOR FUNCTION SUBSTITUTIONS</td>
<td>23</td>
</tr>
<tr>
<td>ID,AINBE</td>
<td>ALLOW INBETWEEN FOR FUNCTION SUBSTITUTIONS</td>
<td>23</td>
</tr>
<tr>
<td>AI</td>
<td>ASSIGN SAME SUBSTITUTION LEVEL</td>
<td>21</td>
</tr>
<tr>
<td>ID,ASYMX,F,1,3</td>
<td>ANTISYMMETRIC FUNCTION IN ARGUMENTS 1,3</td>
<td>26</td>
</tr>
<tr>
<td>&amp;</td>
<td>QUANTITIES TO BE FACTORED OUT</td>
<td>18</td>
</tr>
<tr>
<td>* BEGIN</td>
<td>SEPARATES PROBLEMS, TRANSFER OF COMMON FILES</td>
<td>17</td>
</tr>
<tr>
<td>C</td>
<td>COMMENT CARD</td>
<td>35</td>
</tr>
<tr>
<td>COMMON FN</td>
<td>DECLARE FN COMMON, KEPT OVER * BEGIN</td>
<td>20</td>
</tr>
<tr>
<td>ID,COMP,F</td>
<td>COMPOSE NAMES AND REARRANGE FUNCTION ARGUMENTS</td>
<td>30</td>
</tr>
<tr>
<td>COMMON FN</td>
<td>OPERATOR FOR COMPLEX CONJUGATION</td>
<td>33</td>
</tr>
<tr>
<td>ID,COUNT,O,A,1</td>
<td>ACTION BASED ON WEIGHT OF TERMS</td>
<td>30</td>
</tr>
<tr>
<td>D F(N)=</td>
<td>DATA EXPRESSION, INDEXED X EXPRESSION</td>
<td>14</td>
</tr>
<tr>
<td>D(N1,P)</td>
<td>DELTA FUNCTION OF SPINOR ALGEBRA</td>
<td>31</td>
</tr>
<tr>
<td>DB(N1,N2)</td>
<td>BINOMIAL FUNCTION</td>
<td>32</td>
</tr>
<tr>
<td>DD(A,A)</td>
<td>REPLACE A BY B FOR NUMBERS, DD(N,M)*N/M, DD(N)=1/N</td>
<td>32</td>
</tr>
<tr>
<td>DELETE FN</td>
<td>DELETE COMMON FILES</td>
<td>20</td>
</tr>
<tr>
<td>DF(F,3,...)</td>
<td>REFERENCE TO FROZEN SURFILE</td>
<td>33</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>SWITCH TO INTERACTIVE MDF</td>
<td>37</td>
</tr>
<tr>
<td>DK(N1,N2)</td>
<td>KRONNECKER DELTA FUNCTION</td>
<td>33</td>
</tr>
<tr>
<td>ID,DTPR</td>
<td>SUBSTITUTE INSIDE DOTPRODUCTS</td>
<td>23</td>
</tr>
<tr>
<td>ID,DTPR</td>
<td>SUBSTITUTE INSIDE SPECIFIC VECTOR COMPONENTS</td>
<td>23</td>
</tr>
<tr>
<td>DP(N1,N2,...)</td>
<td>PERMUTATION FUNCTION</td>
<td>33</td>
</tr>
<tr>
<td>DS(J,N,M,E),(Z)</td>
<td>SUMMATION FUNCTION</td>
<td>32</td>
</tr>
<tr>
<td>DT(N1,N2,...)</td>
<td>TETRA FUNCTION</td>
<td>32</td>
</tr>
<tr>
<td>DX(...),*</td>
<td>EXTERNAL FUNCTION VIA SUBROUTINE EXTRA</td>
<td>17</td>
</tr>
<tr>
<td>EN</td>
<td>END OF PROGRAM</td>
<td></td>
</tr>
<tr>
<td>ENTER COMMON</td>
<td>COMMON FILES ARE READ FROM TAPES</td>
<td>21</td>
</tr>
<tr>
<td>EPS</td>
<td>EPSILON FUNCTION, TOTALLY ANTISYMMETRIC TENSOR</td>
<td>31</td>
</tr>
<tr>
<td>ID,EVENX,F,1,3</td>
<td>EVEN FUNCTION IN ARGUMENTS 1,3</td>
<td>26</td>
</tr>
<tr>
<td>ID,EXPAN,FN</td>
<td>REMOVE NF = FROZEN SURFILES</td>
<td>28</td>
</tr>
<tr>
<td>F</td>
<td>FUNCTION NAMELIST</td>
<td></td>
</tr>
<tr>
<td>F A=C</td>
<td>DECLARE A TO BE COMPLEX</td>
<td>15</td>
</tr>
<tr>
<td>F A=I</td>
<td>DECLARE A TO BE IMAGINARY</td>
<td>15</td>
</tr>
<tr>
<td>F A=A</td>
<td>A HAS UNDEFINED PROPERTIES FOR COMPLEX CONJ</td>
<td>15</td>
</tr>
<tr>
<td>* FIX</td>
<td>FIXES QUANTITIES TO BE KEPT OVER * BEGIN</td>
<td>17</td>
</tr>
<tr>
<td>FREEZE FN</td>
<td>SPLIT FILE IN FIXED PART (NF) AND VARIABLE PART</td>
<td>28</td>
</tr>
<tr>
<td>ID,FUNCT</td>
<td>SUBSTITUTE FUNCTION ARGUMENTS</td>
<td>23</td>
</tr>
<tr>
<td>G(J,MU)</td>
<td>GAMMA MATRIX, INDEX MU, LOOP J</td>
<td>31</td>
</tr>
<tr>
<td>G(J)</td>
<td>UNIT GAMMA MATRIX, LOOP J</td>
<td>31</td>
</tr>
<tr>
<td>G5(J)</td>
<td>GAMMAS MATRIX, LOOP J</td>
<td>31</td>
</tr>
<tr>
<td>G6(J)</td>
<td>G5(J)*G5(J)</td>
<td>31</td>
</tr>
<tr>
<td>G7(J)</td>
<td>G5(J)-G5(J)</td>
<td>31</td>
</tr>
<tr>
<td>I</td>
<td>INDEX NAMELIST</td>
<td>15</td>
</tr>
<tr>
<td>I N1=6</td>
<td>INDEX WITH DIMENSION 6</td>
<td>15</td>
</tr>
<tr>
<td>ID</td>
<td>ASSIGN NEXT SUBSTITUTION LEVEL</td>
<td>21</td>
</tr>
<tr>
<td>ID,19</td>
<td>APPLY SUBSTITUTION AT 19 LEVELS</td>
<td>21</td>
</tr>
<tr>
<td>ID,FIGU,20A</td>
<td>TEST ON NUMERICAL COEFFICIENT</td>
<td>28</td>
</tr>
<tr>
<td>ID,FIGRE,20A</td>
<td>TEST ON NUMERICAL COEFFICIENT</td>
<td>28</td>
</tr>
<tr>
<td>ID,IFSHA,20A</td>
<td>TEST ON NUMERICAL COEFFICIENT</td>
<td>28</td>
</tr>
<tr>
<td>INTE</td>
<td>OPERATOR FOR NUMERICAL EVALUATION</td>
<td>34</td>
</tr>
<tr>
<td>KEEP FN</td>
<td>TRANSFER FILES OVER * NEXT</td>
<td>20</td>
</tr>
<tr>
<td>ID,MULTI</td>
<td>KEYWORD FOR A**Nz</td>
<td>22</td>
</tr>
</tbody>
</table>
INDEX | SHORT DESCRIPTION | REFERENCE
--- | --- | ---
N 15 | NUMBER OF DIGITS FOR FLOATING POINT RESULTS | 18
N R | NUMBERS TO BE PRINTED AS RATIO OF 2 INTEGERS | 18
NAME FN | READ IN NAMES OF A COMMON FILE | 16
* NEXT | SEPARATES PROBLEMS, TRANSFER OF KEEP FILES | 17
NPRINT FN | NONPRINT | 19
ID,NUM,A,R,3 | GIVE NUMERICAL VALUE | 29
ID,OMXX,F,1,3 | ODD FUNCTION IN ARGUMENTS 1,3 | 26
OLDNEW A=R | CHANGE OF NAME | 16
ID,ORDER,F,3,II | ORDER FUNCTIONS | 27
ID,ORDERP,P,Q | VECTORS ARE ORTHOGONAL | 27
ID,ORDERP,P,Q | VECTORS ARE ORTHONORMAL | 27
PRINT FN | PRINT ANALYSIS OF INPUT | 19
PRINT CINPUT | DUMP REQUEST | 19
PRINT COUTPUT | DUMP REQUEST | 19
PRINT NSTATISTICS | END INTERACTIVE MODE | 37
PRINT OUTPUT | | 19
PUNCH FN | PUNCH WITH FORTRAN CONTINUATION CHARACTER | 21
R | INPUT OF PREVIOUS RUN | 14
ID,RATIO,X,9,B,C | PARTIAL FRACTION DECOMPOSITION OF XA***X**C | 29
ID,REPLA,A,R,F,1 | REPLACE A BY B AT FIRST ARGUMENT OF F | 26
S | SYMBOL NAMELIST | 15
ID,SPIN1,J | SPINSUMATION, LOOP J | 26
STORE FN | WRITE RESULTS ON TAPE3 | 20
SUM NL | DECLARE NL TO SUMMATION INDEX | 15
ID,SUMXX,F,1,3 | SYMMETRIC FUNCTION IN ARGUMENTS 1,3 | 27
T | END MARK FOR TAPE COPY | 36
TAPE COPY | COPY INPUT TO TAPE3 | 36
TAPE END | INDICATES END OF RESULTS WRITTEN ON TAPE3 | 36
TAPE ENDFILE | WRITE AN END OF FILE ON TAPE3 | 36
TAPE FORWARD | SKIP TAPE3 TILL TAPE END | 37
TAPE NAMENAMES | READ TAPE3 TILL TAPE START | 36
TAPE NAMENAMES | SKIP TAPE3 TILL TAPE START (END OF NAMELISTS) | 37
TAPE PRINT | COPY TAPE3 TO OUTPUT | 39
TAPE PUNCH | COPY TAPE3 TO PUNCH | 36
TAPE READ | USE TAPE3 AS INPUT | 36
TAPE REWIND | REWIND TAPE3 | 36
TAPE START | INDICATES END OF NAMELISTS WRITTEN ON TAPE3 | 36
ID,TTHICK,J,THACE,K | GAMMA AND EPSILON REDUCTION FOR LOOP J | 25
ID,TTHICK,J,THACE,K | TRACES OF LOOP K | 25
UGB | CONJUGATE | 31
UG | SPINOR, UG LOOP, MASS, MOMENTUM FOR SPIN 1/2 | 31
UG | SPINOR, UG LOOP, POLAR, MASS, MOMENTUM FOR SPIN 3/2 | 31
V | VECTOR NAME LIST | 15
WRITE COMMON | COMMON FILES ARE WRITTEN ON TAPE5 | 20
X | SEPARATION OF SETS OF SUBSTITUTIONS | 17
Y | DUMMY FUNCTION ARGUMENT | 11
Z | EXPRESS TO BE EVALUATED | 13
* YEP | SUBSTITUTION APPLIED AT ALL LEVELS | 14
* NEXT | SEPARATES PROBLEMS, TRANSFER OF KEEP FILES | 17
NPRINT FN | NONPRINT | 19
ID,ORDER,F,3,II | ORDER FUNCTIONS | 27
ID,ORDERP,P,Q | VECTORS ARE ORTHOGONAL | 27
ID,ORDERP,P,Q | VECTORS ARE ORTHONORMAL | 27
PRINT FN | PRINT ANALYSIS OF INPUT | 19
PRINT CINPUT | DUMP REQUEST | 19
PRINT COUTPUT | DUMP REQUEST | 19
PRINT NSTATISTICS | END INTERACTIVE MODE | 37
PRINT OUTPUT | | 19
PUNCH FN | PUNCH WITH FORTRAN CONTINUATION CHARACTER | 21
R | INPUT OF PREVIOUS RUN | 14
ID,RATIO,X,9,B,C | PARTIAL FRACTION DECOMPOSITION OF XA***X**C | 29
ID,REPLA,A,R,F,1 | REPLACE A BY B AT FIRST ARGUMENT OF F | 26
S | SYMBOL NAMELIST | 15
ID,SPIN1,J | SPINSUMATION, LOOP J | 26
STORE FN | WRITE RESULTS ON TAPE3 | 20
SUM NL | DECLARE NL TO SUMMATION INDEX | 15
ID,SUMXX,F,1,3 | SYMMETRIC FUNCTION IN ARGUMENTS 1,3 | 27
T | END MARK FOR TAPE COPY | 36
TAPE COPY | COPY INPUT TO TAPE3 | 36
TAPE END | INDICATES END OF RESULTS WRITTEN ON TAPE3 | 36
TAPE ENDFILE | WRITE AN END OF FILE ON TAPE3 | 36
TAPE FORWARD | SKIP TAPE3 TILL TAPE END | 37
TAPE NAMENAMES | READ TAPE3 TILL TAPE START | 36
TAPE NAMENAMES | SKIP TAPE3 TILL TAPE START (END OF NAMELISTS) | 37
TAPE PRINT | COPY TAPE3 TO OUTPUT | 39
TAPE PUNCH | COPY TAPE3 TO PUNCH | 36
TAPE READ | USE TAPE3 AS INPUT | 36
TAPE REWIND | REWIND TAPE3 | 36
TAPE START | INDICATES END OF NAMELISTS WRITTEN ON TAPE3 | 36
ID,TTHICK,J,THACE,K | GAMMA AND EPSILON REDUCTION FOR LOOP J | 25
ID,TTHICK,J,THACE,K | TRACES OF LOOP K | 25
UGB | CONJUGATE | 31
UG | SPINOR, UG LOOP, MASS, MOMENTUM FOR SPIN 1/2 | 31
UG | SPINOR, UG LOOP, POLAR, MASS, MOMENTUM FOR SPIN 3/2 | 31
V | VECTOR NAME LIST | 15
WRITE COMMON | COMMON FILES ARE WRITTEN ON TAPE5 | 20
X | SEPARATION OF SETS OF SUBSTITUTIONS | 17
Y | DUMMY FUNCTION ARGUMENT | 11