

## Appendix 5

### A.5 CSMP, Continuous System Modeling Program

H.H. van Laar and P.A. Leffelaar

#### A.5.1 Introduction

CSMP stands for Continuous System Modeling Program, version III. It has been extensively described in the Program Reference Manual by IBM (SH19-7001-3, 1975). CSMP is a specific dynamic simulation language used to integrate rate equations to obtain the state of the model as a function of time. CSMP is a non-procedural language, which means that the user can write programs in a conceptual order and CSMP will sort the statements in a computational order. An important feature in CSMP is the availability of numerical integration routines which are easy to use. Moreover, CSMP automatically keeps track of time in dynamic simulation. CSMP provides special functions (e.g. interpolation), and as its source program is written in FORTRAN (FORMula TRANslation), the researcher may use FORTRAN statements as well as all FORTRAN library functions in more advanced models. Tabular and/or graphical output can be obtained by just listing the variables on a special label.

#### A.5.2 The structure of the model

One starts a program with a TITLE label containing a short identification of the program. In a CSMP program, 3 segments can be distinguished: INITIAL,

TITLE CSMP STRUCTURE	
INITIAL	sorting
.....	
NOSORT	no sorting
.....	
SORT	sorting
.....	
DYNAMIC	sorting
.....	
NOSORT	no sorting
.....	
SORT	sorting
.....	
TERMINAL	no sorting
.....	
END	
STOP	
ENDJOB	

Figure A.1. General layout of a CSMP program. Sorting: sorted by CSMP. No sorting: must be sorted by the user.

**DYNAMIC** and **TERMINAL**. These statements (labels) indicate that the computations must be performed before, during and after a simulation run, respectively (Figure A.1). If one is using these segments, then each segment label closes the former segment. To close the entire program, one must use the statements **END**, **STOP** and **ENDJOB**, respectively, each on a separate line, and **ENDJOB** must start in the first column.

In the **INITIAL** segment, computations are executed only once per run. The use of the segment is optional. The **INITIAL** segment can be used for computing the results which are used as input data for the dynamic section of the program. All initial conditions and parameters can be given values in this segment. The segment where the simulation takes place is the **DYNAMIC**. This segment is normally the most extensive one in a model. It contains the complete description of the model dynamics, together with any other computation required during the simulation. For some models, the program consists of just the **DYNAMIC** segment. The segment may be declared explicitly by the label **DYNAMIC**, but if there is no **INITIAL** or **TERMINAL** segment, the **DYNAMIC** label can be omitted. The **TERMINAL** segment can be used for computations required at the end of the run, after completing the simulation. This can be a calculation based on the final values of one or more variables. As in the **INITIAL** segment, the computations are executed only once. Also the **TERMINAL** segment is optional.

**CSMP** is provided with a sorting algorithm to free the user from the task of correctly sequencing the statements. The user can then focus his attention on defining the problem, and can put the statements in a conceptual order. The statements will be put in computational order during the translation phase, and are then written into **UPDATE**, a subroutine created by the **CSMP** compiler which contains the structure of the model with sorted **FORTRAN** statements. All statements in the **INITIAL** and **DYNAMIC** segments are placed in computational order. The statements in the **TERMINAL** segment, however, must be sorted by the user.

It is sometimes desirable for statements in the program not to be sorted by the compiler, for instance where branching conditions are desired, or a decision must be made. In that case the statements, written either in **CSMP** or **FORTRAN**, must stay in a fixed order. There are two ways in **CSMP** to avoid sorting the statements. The simplest method is to divide the **INITIAL** and **DYNAMIC** segments into sections by means of the labels **NOSORT**, which prevents the statements from being sorted, and **SORT**, which ends a **NOSORT** section. Thus, the program will be split into individual blocks. The program blocks between the **NOSORT-SORT** sections will now be sorted individually, and during the translation phase, statements from one **SORT** section cannot be moved to another **SORT** section. Since this usually gives computational problems, it is recommended to write full sections in **NOSORT** or to use **PROCEDURES**. **FORTRAN** statements must always be sequenced in computational order and can be defined only in a non-sortable section.

A more elegant method of defining sections which are not sorted by the CSMP compiler, but which may be sorted as a whole, is to define PROCEDURES. The PROCEDURE is treated as an entity. It is sorted as a functional block on the basis of input and output names given by the user when defining the PROCEDURE. These names correspond with those of the statements in the functional block. A PROCEDURE is defined as:

```

PROCEDURE OUTP1,OUTP2=NAME(INPUT1,INPUT2)
....
....
....
....
ENDPROCEDURE

```

} CSMP and/or FORTRAN  
} statements  
} sorted by the user

} sorted by  
} CSMP

The statements describing the PROCEDURE are placed between the statements labelled PROCEDURE and ENDPROCEDURE and are not sorted internally. Variables defined within a PROCEDURE block and not appearing in the definition, e.g. because they are not needed to sort the block, are not available for data output by means of PRINT or OUTPUT (see also section 'Labels'). If these variables are needed for output, they should be included as output names in the PROCEDURE definition. A PROCEDURE is a block of sequenced statements which is executed where defined. Contrary to PROCEDURES, SUBROUTINES can be called for more than once in a program.

SUBROUTINES are another way of structuring programs. SUBROUTINES can be called from within the CSMP program, but are defined between the labels STOP and ENDJOB. The simplest way to use SUBROUTINES will be discussed below. A SUBROUTINE is called for by the sortable statement:

OUT1,OUT2=SUBNAM(IN1,IN2,IN3) Equation (A.2)

where the variables on the left hand side of the equals-sign are the results of the calculations performed in the SUBROUTINE named SUBNAM. (Note that there must be at least two variables on the left hand side, otherwise the translator accepts SUBNAM as a FUNCTION; for details see below.) The input variables for these calculations are listed between brackets following the name of the SUBROUTINE. The variable names, either referring to reals, integers or arrays, serve different purposes: (1) they are used to sort the statement within the CSMP program, as in the case of a PROCEDURE, (2) they attend to the communication between the CSMP program and the SUBROUTINE. The CSMP compiler interprets Equation A.2 as:

CALL SUBNAM(IN1,IN2,IN3,OUT1,OUT2)

which is placed in UPDATE. Thus, the output variables are placed just after the input variables without changing their sequence. Statements placed between STOP and ENDJOB are not processed by the CSMP compiler. Thus SUBROU-

TINE definitions are directly placed into UPDATE, and the arguments in the definition must agree with the call to the SUBROUTINE as generated by the CSMP translator:

```
SUBROUTINE SUBNAM(IN1,IN2,IN3,OUT1,OUT2)
```

The positions of the variable names in the call statement and in the definition of the SUBROUTINE correspond to one another, but need not have the same names. However, corresponding places in the list of variables must contain the same type of variable.

All the usual FORTRAN rules apply to SUBROUTINES. Some important rules are:

1. Statements begin in column 7 or higher; numbers of continuation labels must be placed in the first 5 columns; column 6 is reserved to indicate whether the line is a continuation of the preceding one by, for example, placing a dollar (\$) sign there.
2. Variables beginning with I, J, K, L, M or N are considered integer, while others are considered real. This may be cumbersome, as in CSMP all variables are considered real except those placed on the FIXED label. It is good practice to apply this rule to FORTRAN as well by introducing the statements:

```
    IMPLICIT REAL(A-Z)
    INTEGER ...
```

directly after the line which defines the SUBROUTINE. The first statement declares all variables beginning with A up to and including Z real; subsequently, the specific variables listed after the INTEGER label are declared integer. The INTEGER label in FORTRAN is equivalent to the FIXED label in CSMP.

3. Array variables in SUBROUTINES need memory storage:  
 DIMENSION A(100),...  
 Here, 100 locations are reserved for the variable A. The DIMENSION label in FORTRAN is equivalent to the STORAGE label in CSMP.
4. More than one RETURN statement may be used to return to the place in the CSMP program where the SUBROUTINE was called. However, a SUBROUTINE is always terminated by the statements:

```
    RETURN
    END
```

The general layout of a SUBROUTINE is shown in Figure A.2.

In the call of a SUBROUTINE in CSMP, at least two variables must be listed on the left hand side of the equals-sign. If there is only one variable listed, the CSMP compiler interprets the statement as a so-called function subroutine, where the result of the calculation is returned to the calling program using the name of the subroutine rather than the output variable. This implies also that only a single variable can be output from a function subroutine, while often array

column

1234567

```
SUBROUTINE SUBNAM(NLOC,IN1,IN2,IN3,INTB,OUT1,OUT2)
IMPLICIT REAL(A-Z)
INTEGER NLOC, ....
DIMENSION ....
.
.
OUT1 = AFGEN(NLOC,INTB,IN1)
.
.
RETURN
END
```

Figure A.2. General layout of a SUBROUTINE.

results are needed, and that can only be achieved by arguments in subroutines. The function subroutine is not discussed here, but when there is one output variable in the subroutine, misinterpretation by the CSMP compiler is avoided by including a dummy variable in the list of output variables. This dummy variable must also be given in the definition.

Many CSMP functions may be used within SUBROUTINES, but specifically excluded is the INTGRL function. The use of functions such as LIMIT, EXP and SQRT is similar to their use in CSMP. When functions starting with I, J, K, L, M or N are used (e.g. INSW or LIMIT, which are examples of function subroutines) the name needs to be declared real either by:

```
REAL INSW, LIMIT
```

or implicitly as explained above. History functions require both past and present values of the inputs to calculate their outputs. When history functions, e.g. AFGEN or NLFGEN, are used in SUBROUTINES, storage locations must be indicated. This is done in the CSMP program by:

```
HISTORY SUBNAM(5)
```

at the beginning of the program. The number '5' corresponds to the number of storage locations for an AFGEN function, which is used in SUBROUTINE SUBNAM. For the use of other functions, reference is made to the CSMP manual. The function which is interpolated by AFGEN is transferred to the SUBROUTINE by including its name in the list of input variables. The combination of the HISTORY label and the call for the SUBROUTINE SUBNAM:

```
OUT1,OUT2=SUBNAM(IN1,IN2,IN3,INTB)
```

where INTB is the function name, causes the generation of the following UP-

```

column
1234567
      SUBROUTINE SUBNAM(IN1,IN2,IN3,OUT1,OUT2)
      IMPLICIT REAL(A – Z)
      INTEGER ....
      DIMENSION ....
      .
      .
      IF(.....) RETURN
      .
      .
10    CONTINUE
      .
      .
      RETURN
      END

```

Figure A.3. General layout of a SUBROUTINE containing a history function.

DATE statement:

```
CALL SUBNAM(1,IN1,IN2,IN3,INTB,OUT1,OUT2)
```

The first argument between brackets, e.g. 1, contains a number corresponding with the first storage location assigned to the SUBROUTINE. The SUBROUTINE definition must agree with the call generated by the CSMP compiler. The general layout of a SUBROUTINE containing an AFGEN function is shown in Figure A.3. The integer variable NLOC (Figure A.3) takes the value assigned by the CSMP compiler in the SUBROUTINE call in UPDATE. Figure A.4 summarizes all this. This demonstration program simulates the change in the amount of water in two lakes connected in series when an instantaneous doubling of water inflow occurs.

### A.5.3 Some elements of CSMP

*Numeric constants* There are two types of constants: integer and real. Integers are whole numbers with a maximum of 10 digits without a decimal point. Real (floating-point) constants are numbers written with a decimal point, with a maximum of 7 digits. A real constant may be followed by a decimal exponent written as the letter E, followed by a signed or unsigned one or two digit integer constant. The decimal E format forms a real constant that is the product of the real constant portion multiplied by 10 raised to the desired power; e.g.  $213.15 = 2.1315E2 = 2131.5E-01$ . Real constants are restricted to a total of 12 characters.

Figure A.4. Example of the use of subroutines in CSMP.

```

TITLE Demonstration program on how a subroutine can be used
INITIAL
HISTORY INICON(5), FLOWS(5)
FIXED N
STORAGE FLOW(3)
***** Defenition of parameters *****
PARAM DTSYS=8., N=2

***** Timer variables and integration method *****
TIMER FINTIM=14., OUTDEL=2., DELT=2.
METHOD RECT

***** Output results *****
OUTPUT H(1), H(2)

***** Function defining inflow rate into the first reservoir
FUNCTION INTB = 0.0,100., 1.99,100., ...
                2.0,200., 14.0,200.

***** Initial calculations *****
REALN =N
TC     =DTSYS/REALN

***** Initial conditions of reservoirs *****
IH,DUM1      =INICON(N,INTB,TIME,TC)

*****
*****
DYNAMIC
H             =INTGRL(IH,NETFLO,2)

***** Net flow for each reservoir *****
NETFLO,DUM2   =NETFLS(N,FLOW)

***** Individual flows into and out of each reservoir *****
FLOW,DUM3     =FLOWS(N,INTB,TIME,TC,H)

END
STOP
*****
***** Subroutines called from initial *****
*****

SUBROUTINE INICON(NLOC,N,INTB,TIME,TC,
$           IH,DUM1)
IMPLICIT REAL(A-Z)
INTEGER I,N,NLOC
DIMENSION IH(2)
IN       =AFGEN(NLOC,INTB,TIME)
DO 10 I =1,N
    IH(I)=IN*TC
10 CONTINUE
RETURN
END

```

\*\*\*\*\*  
 \*\*\*\*\* Subroutines called from dynamic \*\*\*\*\*  
 \*\*\*\*\*

```

SUBROUTINE NETFLS(N, FLOW,
$              NETFLO, DUM2)
  IMPLICIT REAL(A-Z)
  INTEGER      I, N
  DIMENSION   FLOW(3), NETFLO(2)
  DO 10 I = 1, N
    NETFLO(I) = FLOW(I) - FLOW(I+1)
10 CONTINUE
  RETURN
  END

```

\*\*\*\*\*

```

SUBROUTINE FLOWS(NLOC, N, INTB, TIME, TC, H,
$              FLOW, DUM3)
  IMPLICIT REAL(A-Z)
  INTEGER      I, N, NLOC
  DIMENSION   H(2), FLOW(3)
  IN          = AFGEN(NLOC, INTB, TIME)
  FLOW(1) = IN
  DO 10 I = 2, N+1
    FLOW(I) = H(I-1) / TC
10 CONTINUE
  RETURN
  END

```

\*\*\*\*\*  
 ENDJOB

*Variables* The name of a variable can contain one to six characters and the first character must be a letter. No blanks or special characters (e.g. +,\*(-:/).) are allowed. For so-called 'reserved words' one is referred to the Reference Manual. All names of labels, functions and data statements are reserved words. In CSMP programs, before the label STOP all variables are declared real. When using integer variables in a program, these variables should be explicitly declared at the beginning of the program by the label FIXED.

*Operators* As in FORTRAN, the operators and the order in which operations are performed are:

( )	grouping of variables and/or constants	1st
**	exponentiation	2nd
*	multiplication	} 3rd
/	division	
+	addition	} 4th
-	subtraction	
=	replacement	5th



Functions and expressions within parentheses are always evaluated first. For operators of the same order, the component operations of the expression are performed from left to right. There is an exception for exponentiation, where the evaluation is performed from right to left. Thus, the expression  $A^{**}B^{**}C$  is evaluated as  $A^{**}(B^{**}C)$ .

*Functions* A description of various CSMP and FORTRAN functions is given in Tables A.1 and A.2. Extra attention will be given here to the INTGRL and AFGEN functions. The general instruction to execute computations with relation to numerical integration is:

$$A = \text{INTGRL}(IA, \text{RATE})$$

in which A is the output of an integrator, RATE is the rate of change of A, which is integrated over time, and IA is the initial condition of A in a simulation run. Initial conditions are introduced using an INCON label. The feature of integrator arrays is very useful when simulating spatially distributed systems, e.g. gas diffusion through water, or dispersion in time (exponential delays), where space or time is divided into layers or classes. Each class or layer is represented by an integrator and, between layers, transport takes place (Figure A.4 gives an example of the use of an integrator array of length 2). One can use array integrals with subscripted variables for computations that have a similar structure. The statement:

$$A = \text{INTGRL}(IA, \text{RATE}, 10)$$

specifies an array of 10 integrators in which A(I), IA(I) and RATE (I) are, respectively, the output, initial conditions and input variables. The third argument, 10, is an integer constant. The initial conditions of an integrator array must be given by means of a TABLE statement, e.g.

$$\text{TABLE } IA(1-10) = 10 * 5.$$
$$\text{TABLE } IA(1-10) = 2., 4., 6., 3., 7., 1., 4 * 0.$$

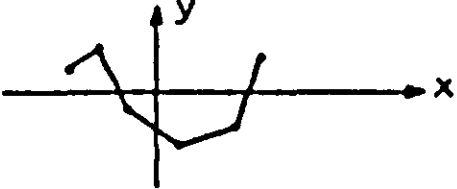
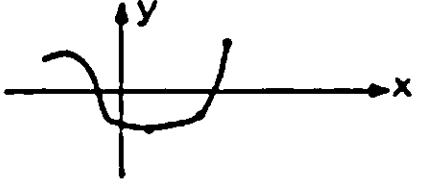
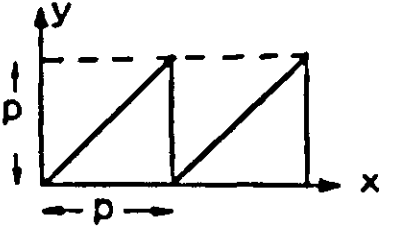
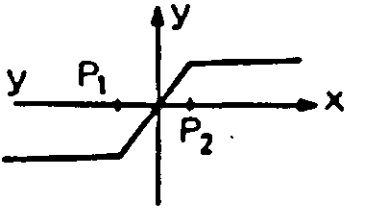
In the latter TABLE statement, the first 6 numbers represent real numbers, the '4' represents an integer number to indicate that the last four numbers have the same value (e.g. zero). For subscripted variables, one usually needs to reserve memory storage. For variables in integrator arrays, however, this happens automatically. The computer system generates:

$$\text{STORAGE } A(10), IA(10), \text{RATE}(10)$$

The rates RATE(1) to RATE(10) can be computed in a DO loop:

```
DO 100 I = 1,10
  RATE(I) = .....
100 CONTINUE
```

Table A.1. Some CSMP functions.

CSMP III Functions	Equivalent Mathematical Expression
Integrator $Y = \text{INTGRL}(IC, X)$ where: $IC = y_{t_0}$	$y(t) = \int_{t_0}^t x dt + y(t_0)$ where: $t_0 = \text{start time}$ $t = \text{time}$
Arbitrary function generator (linear interpolation) $Y = \text{AFGEN}(\text{FUNCT}, X)$	$y = f(x)$ 
Arbitrary function generator (quadratic interpolation) $Y = \text{NLFGEN}(\text{FUNCT}, X)$	$y = f(x)$ 
Modulo function $Y = \text{AMOD}(X, P)$	$y = x - nP$ n is an integer value such that $0 \leq y < P$ 
Limiter $Y = \text{LIMIT}(P1, P2, X)$	$y = P_1 ; x < P_1$ $y = P_2 ; x > P_2$ $y = x ; P_1 \leq x \leq P_2$ 
Not $Y = \text{NOT}(X)$	$y = 1 \text{ if } x < 0$ $y = 0 \text{ if } x > 0$
Input Switch Relay $Y = \text{INSW}(X1, X2, X3)$	$y = x_2 \text{ if } x_1 < 0$ $y = x_3 \text{ if } x_1 \geq 0$

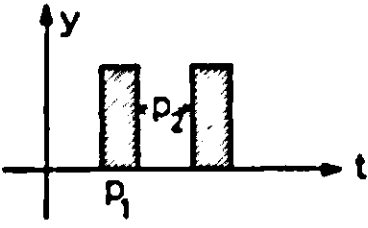
in which 100 is the 'name' of the DO loop, I is a counter which takes the values 1, 2, 3, ... 10, successively so that all RATEs will be computed. I should be an integer variable; therefore, it must be declared integer by the statement FIXED I. A DO loop cannot be sorted by CSMP, so it must be put in a NOSORT-SORT section, a PROCEDURE or in a SUBROUTINE.

The INTGRL statement merely indicates that the specified RATEs have to be integrated. The numerical integration method has still to be specified. This is done on the METHOD label, e.g. METHOD RECT, which causes rectangular integration to take place.

The Arbitrary Function GENERator (AFGEN function) in CSMP interpolates linearly between supplied points (e.g.  $x_1, y_1$  and  $x_2, y_2$ ). The function value  $\hat{y}$  for a certain  $x$  is then calculated by the expression:

$$y = y_1 + (x - x_1) \cdot (y_2 - y_1) / (x_2 - x_1)$$

Table A.1. Continued.

<p>Dead time (DELAY)</p> <p>Y=DELAY(N,P,X)</p> <p>where: P=delay time N=number of points sampled in interval p (integer constant) and must be <math>\geq 3</math>, and <math>\leq 16,378</math></p>	<p><math>y=x(t-p) ; t &gt; p</math></p> <p><math>y=0 ; t &lt; p</math></p> <p>Equivalent Laplace Transfer Function:</p> $\frac{Y(s)}{X(s)} = e^{-ps}$
<p>Implicit function</p> <p>Y=IMPL(IC,P,FOFY)</p> <p>where: IC=first guess P = error bound FOFY=output name from final statement in algebraic loop definition</p>	<p><math>y = f(y)</math></p> <p><math> y - f(y)  \leq p  y </math></p>
<p>Impulse generator</p> <p>Y=IMPULS(P1,P2)</p> <p>where: P1=time of first pulse P2=interval between pulses</p>	<p><math>y=0 ; t &lt; p_1</math></p> <p><math>y=1 ; (t-p_1) = kp_2</math></p> <p><math>y=0 ; (t-p_1) \neq kp_2</math></p> <p><math>k = 0, 1, 2, 3, \dots</math></p> 

and is written in CSMP:  $Y = \text{AFGEN}(\text{XYTB}, X)$

FUNCTION XYTB = (0.,0.),(2.,0.4),(6.,0.1)

where the first number of a pair represents the variable x; the values of x must increase monotonically. In most models, linear interpolation suffices. For higher order interpolation methods, reference is made to the CSMP manual.

#### A.5.4 Labels

Output control statements:

- TITLE** allows the user to identify the program, and the title appears on top of each page of the output listing;
- PRINT** is used to specify upto 55 variables whose values will be printed at each PRDEL interval in a tabular form (only real and not integer variables!). The PRINT label can be used only once in a CSMP program, as a second label would override the first;
- OUTPUT** is used to obtain printed output together with graphical output of

Table A.2. Some FORTRAN functions, which can be used in CSMP statements.

FORTRAN Functions	Equivalent Mathematical Expression
Exponential Y = EXP ( X )	$y = e^x$
Trigonometric sine (argument in radians) Y = SIN ( X )	$y = \sin (x)$
Trigonometric cosine (argument in radians) Y = COS ( X )	$y = \cos (x)$
Square root Y = SQRT ( X )	$y = \sqrt{x}$
Largest value (Real arguments and output) Y = AMAX1 ( X1, X2 )	$y = \max (x_1, x_2)$
Smallest value (Real arguments and output) Y = AMIN1 ( X1, X2 )	$y = \min (x_1, x_2)$

upto 5 variables at each OUTDEL interval. When the number of OUTPUT variables exceeds 5, graphical output is suppressed and printed output (of upto 55 variables) is given alone.

PRINT is used to organize the graphical OUTPUT.

A useful option is the following example:

```
PAGE GROUP, NTAB = 0, WIDTH = 80
OUTPUT A(1), A(3)
OUTPUT A(1), A(10)
PRINT A(1), A(3), A(10)
```

Here the graphical output of A(1) and A(3), and A(1) and A(10) may easily be compared as these variables are plotted on the same scale (PAGE GROUP). On the plots no tabular output is given (PAGE NTAB = 0), so as to obtain the largest possible graphs.

Tabular output for these variables is obtained by the PRINT statement.

The statement PAGE WIDTH = 80 causes the graphs to be inspected on the computer terminal. A PAGE statement preceding all the OUTPUT statements applies to the whole group. A PAGE statement that follows an OUTPUT statement is assigned to that statement.

Execution control statements:

Values of timing variables are specified with the TIMER statement:

FINTIM final value of time for terminating a simulation;

OUTDEL time interval for graphical output;

PRDEL time interval for tabular output;

DELT integration interval;

TIME initial value of time, to be specified only if not zero.

Example: TIMER FINTIM = 100., PRDEL = 5., OUTDEL = 5.  
TIMER DELT = 1.  
TIMER TIME = 10.

METHOD identifies the desired integration routine. Note that if the integration method is not specified, the integration routine RKS is default.

Example: METHOD RECT

END completes the specifications of the model;

STOP terminates the simulation run;

ENDJOB terminates the job.

If a simulation is to be repeated with new data and/or execution control statements, the statements are to be placed between two END statements:

Example: ....  
PARAM A = 10.  
TIMER FINTIM = 50.  
END  
PARAM A = 20.  
TIMER FINTIM = 100.  
END  
STOP  
ENDJOB

Data statements:

PARAM assignment of a numeric value to a parameter;

CONST assignment of a numeric value to a constant;

INCON assignment of a numeric value to an initial condition;

TABLE assignment of numeric values of (initial) constants to arrays;

FUNCTION assignment of the numeric relation between two variables.

Example: PARAM A = 10., B = 20.  
CONST PI = 3.1415  
INCON IA = 0., IB = 5.  
TABLE A(1-5) = 1., 2., 3., 4., 5.  
FUNCTION TEMPTB = (0., 0.), (10., 1.), (40., 1.5)

### A.5.5 Syntax

Some syntax rules may be helpful to make programs more readable:

- try to split up your program into an INITIAL, a DYNAMIC and, if necessary, a TERMINAL segment. Remember that statements in a TERMINAL segment must be sequenced by the user;
- lump all parameter specifications at the beginning of your program, so as to have a better overview of them;
- place all INTGRL statements together, e.g. just at the beginning of the DYNAMIC;
- lump all FUNCTIONS that remain unchanged between simulation runs at the end of the CSMP part of your program before the label END. Other FUNCTIONS can be put near the parameter specifications at the beginning of the program.
- start CSMP statements in the first column;
- start FORTRAN statements in the seventh column;
- make short comments in your program in the proper place;
- use \*\*\*\*\* to begin comments in CSMP;
- use C\*\*\*\*\* to begin comments in FORTRAN subroutines;
- use blank lines and \*\*\*-lines to distinguish between different program parts;
- use blanks (spaces) to line up, e.g. equals-signs (=) and to distinguish between pairs of data in FUNCTION statements;
- continue CSMP statements on the next line by typing three dots (...) on the line to be continued;
- continue FORTRAN statements in subroutines by typing a dollar sign (\$) in the sixth column of the line following the line which is to be continued;
- when the history function AFGEN is used within a subroutine, the user should reserve 5 storage locations using the label HISTORY SUBNAM(5) in the CSMP program. Here, SUBNAM stands for the name of the subroutine where the AFGEN is to be used. If AFGEN is used in the CSMP program itself, memory is automatically allocated by the translator .