IBM CONFIDENTIAL

ADVANCED COMPUTING SYSTEMS
SAN JOSE
March 15, 1966

MEMORANDUM TO: File
SUBJECT: ACS Simulation Technique

D. P. Rozenberg

L. Conway

R. H. Riekert

LC/DP/RR:kp

cc: Architectural Distribution List

022

INTRODUCTION

A brief description of computer simulation of physical systems in general and the features of current simulation languages is given.

A technique is then described for simulation using FORTRAN IV, which maintains the essential features of current simulation languages with a great improvement in run times and core requirements.

This technique may be useful in the production of very large simulation programs where run times and core requirements are such that programming in existing simulation languages may not be feasible.

SYSTEM SIMULATION

Assume that it is of interest to study the behavior of complex systems or automata. If the level of complexity is such that the number of states of the system and the possible sequences of states is very large, then a logical approach to such a study is to simulate the system using a digital computer.

Physical systems are usually described in terms of laws or logical rules relating causes and effects; i.e., a given state together with inputs to the system causes or determines the state (or the probability of selecting the state) at some future time. The "behavior" of the system is thus the sequence of states of the system during the passage of time, in response to a specific input sequence.

A computer simulation thus consists of identifying variables which determine the states of a system and the rules for future state selection (the cause and effect relation) and implementing this model with a computer program. Thus it is possible to artificially experiment with the system, and to study the sequence of states for chosen sequences of inputs, with time as an independent variable of the simulation.

In simulating a system it is necessary to perform a computation only at those times when a state or an input has changed since it is only at such times that a future change of state can be caused. It is therefore not necessary to examine the system at regular clocked intervals. Indeed, this may be vastly more efficient, than examining a system at clocked intervals of simulated time if the time interval between changes of state varies over a wide range of values.

Thus it is found that digital simulation languages may provide the programmer with utility routines for (1) providing a means of causing at future times those effects determined by past and present system states and inputs, and (11) advancing time, as an independent variable of the simulation, to the next scheduled effect (change of state) or to the next change of the input sequence, and (111) passing control to that subroutine which simulates the effect and which itself may cause future effects. Perhaps the best known simulation languages of this type are SIMSCRIPT and GPSS . These languages have in addition to the above features a number of utilities which (1) ease the specification of variables and events,(2) ease the writing of the simulation model description,and (3) simplify the production of output routines.

024

For many purposes these additional utilities are not essential. Indeed, they may cost a high overhead in terms of core space and running time.

GPSS has eased the writing of the simulation to the point where one often cannot specify sufficiently complex tests for branching. Thus, it does not document well a complex description. SIMSCRIPT is sufficiently general but a high cost is extracted in storage and running time because it attempts to simplify the handling of variables.

So, to have a powerful simulation language or technique without all the unnecessary utility features of existing languages, it was decided to write utility routines to perform the basic simulation requirements. A decision had to be made on the language in which to write the simulator utilities routines and also the simulated system description.

If it is important to use the program listings as documentation of the model, a high level language may be necessary. Otherwise, an assembly language might give slight time and storage advantages. In either case, it is desirable to use a common language which runs under a reasonably powerful operating system.

Since in most detailed simulations, the exact model description and documentation can only reside in the simulation program listings, a high level language was chosen as the basic language.

Thus, the utility routines described in the following sections and the model descriptions are all written in FORTRAN IV which is a common high level language running under IBSYS. IBSYS is an operating system which is sufficiently powerful so as to be a valuable aid in running and debugging programs.

THE FORTRAN IV SIMULATION ROUTINES

A general description of the simulation utility routine written in FORTRAN IV follows.

The central idea in the operation of the simulation program is the ordered placement of event notices into a calender of future events as the related cause statements are encountered. The calender is ordered according to increasing time of occurrence. When an event terminates (i.e., the event subroutine terminates), the ordering of the calender indicates the most imminent event and its scheduled time of occurrence. Thus time can be advanced to that scheduled time and the appropriate event subroutine can be called.

A set of arrays, located in blank common, comprise the calender. An event notice consists of one element from each array with the same index. Each notice contains linking information, the scheduled time of occurrence, an indication of the event routine to be called, and three parameters, to be used by the event routine. An event notice will be said to occupy a row of the calender.

During the execution of an event routine, conditions may call for the causing of an event. This is implemented by calling utility subroutine CAUSE with the parameter set: Name of event routine being caused, the time at which the event is to occur, and zero to three parameters to be passed to the event routine. The utility subroutine CAUSE will place in the calender the appropriate event notice. An event may cause any number of events including itself to occur at a future time.

After completion of an event routine,,control is returned to routine MAIN. MAIN then calls the utility TSTEP which extracts the next most imminent event from the calender, sets simulated time to the scheduled time of that event, and transfers control to the appropriate event routine. Upon completion of one event routine, control is passed to next most eminent event routine which will then have the capacity for causing additional events.

In some instances it is desirable to cancel an event which may have been scheduled for the future. To accomplish this a utility subroutine REMOVE is included. It is called with the name of the event to be canceled as a parameter and its function is to search the calender for the first instance of an event notice having the name of the event to be canceled. That event notice is then removed.

The routine package for any given simulation would contain MAIN, CAUSE, REMOVE, and TSTEP plus all of the event subroutines necessary for specifying the model being simulated. CAUSE, REMOVE, and TSTEP are all utility subroutines which are invariant from one simulation to another. MAIN varies from one simulation to another only in that it is desirable to have MAIN contain common statements which include all the systems variables and initializations of system variables.

Included in COMMON are the special variables and the system variables. The special variables include the calender arrays; TIME-the current value of simulated time; IPAR 1, IPAR 2, and IPAR 3 - the parameters associated with the current event; and ISL and ITL - pointers utilized in the calender manipulation. The system variables are those variables in terms of which the programmer describes his simulation model.

The calender consists of six single indexed arrays which are indexed by the same pointer. Thus the calender will be referred to as though it were a two dimensional array with six columns. Column 1 contains linkage for the ordering of event notices. Column 2 contains the time of occurrence while Column 3 contains the reference to the event routine. The remaining columns contain parameters to be passed to the event routine; associated with the event are two pointers - ITL which specifies the next most imminent event and ISL which specifies the row to be filled by the next call of subroutine CAUSE.

As part of the initialization in MAIN, the linkage in the calender is set up. The first row is linked to the second, the second to the third, and so forth. The link in the last row is set to zero to indicate the end of the chain. The first row of the calender is set to indicate an event with a very large value of scheduled time. (This simplifies the calender searching in CAUSE. Finally, ITL is set to 1 and ISL is set to 2.

To schedule an event (i.e., place an event notice in the calender) the time of occurrence, the event routine reference, and the three input parameters are stored in positions 2 through 6 of the row indexed by ISL. Following this, ISL is set equal to the value of the link in the same row. Next, column 2, the time of occurrences, is searched beginning with the row designated by ITL in the order given in column 1, the linkage column. The object of that search is to find an event row k with a time of occurrence which is greater than the occurrence time of the event being scheduled. When such a row is found, the links are adjusted to schedule the new event ahead of the event in row k. The initial event in row 1 guarantees that we find a row k.

Whenever TSTEP is called, position 2 is stored in the COMMON variable TIME, and positions 4, 5, and 6 are stored variables IPAR 1, IPAR 2, and IPAR 3. In addition, the old value of position 1 goes into ITL, the old value of ITL goes into ISL, and the old value of ISL goes into position 1. Finally, the event routine reference is used to call the appropriate event.

An example will now be given to illustrate calender manipulation. Assume that the calender is in the state given in figure 1.

027

	Calender					
Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	4	1.0	Event 1			
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	5	3.0	Event 5			
7	8					
8	9					
9	10					

ISL=7, ITL=2

FIGURE 1

Assume that the next encountered utility call is

```
     CALL   CAUSE   (EVENT 12, 3.25, 0, 0, 0).
```

The result is shown is figure 2.

Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	4	1.0	Event 1			
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	7	3.0	Event 5			
7	5	3.25	Event 12			
8	9					
9	10					

ITL=2,   ISL=8

FIGURE 2

If the next encountered utility call is:

```
    CALL   TSTEP
```

The result is given in figure 3.

Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	8					
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	7	3.0	Event 5			
7	5	3.25	Event 12			
8	9					
9	10					

ITL=4,   ISL=2

FIGURE 3

030

The final subject of this section is the transfer of control to the intended event subroutine when the statement CALL TSTEP is encountered in MAIN. Two satisfactory methods have been used. The first method utilizes FORTRAN IV in a perfectly straight forward manner and is the method to be described in this report. The other method (Method 2) has the advantage of being simpler and easier to use than Method 1, but has the disadvantage of depending on specific characteristics of the IBM 7090/94 IBSYS compiler.

In using Method 1 a variable in a block of named common is defined for each event routine. This variable is the event reference mentioned earlier and is thought of as the event name while the event subroutine name consists of the same FORTRAN N symbol prefixed with an X. For example, a particular simulation model might consist of the following five events. The corresponding subroutine names are also given below.

Event Names	Subroutine Names
MOVE	X MOVE
GENER	X GENER (A)
DELAY	X DELAY
PROC	X PROC (X,Y,Z)
FINIS	X FINIS

Further, it is required that the event names be assigned unique integer values from 1 thru N where N is the number of events. The initialization of event names may be done in routine MAIN.

The organization of MAIN could be as follows:

```
      COMMON 11      . . .
      COMMON /NAMES/     MOVE, GENER, DELAY, PROC, FINIS

      INTEGER, GENER, DELAY, PROC, FINIS

C     SYSTEM INITIATION STATEMENTS

C     CALENDER INITIALIZATION STATEMENTS

      MOVE = 1
      GENER = 2
      DELAY = 3
      PROC = 4
      FINIS = 5
```

031

```
C    PLACE INITIAL EVENT NOTICE
     CALL CAUSE (MOVE, 1.0, 0, 0, 0)

1000 CALL TSTEP (NEVENT)
     GO TO (1, 2, 3, 4, 5), NEVENT

   1 CALL X MOVE
     GO TO 1000

   2 CALL X GENER (IPAR 1)
     GO TO 1000

   3 CALL X DELAY
     GO TO 1000

   4 CALL X PROC (IPAR 1, IPAR 2, IPAR 3)
     GO TO 1000

   5 CALL X FINIS
     GO TO 1000

     END
```

Thus TSTEP returns as the event reference the event number defined in the initialization of event names. The event number is then used to branch to the statement which calls the intended event subroutine.

Method 2 requires less bookkeeping on the part of the programmer. The event subroutine names are the same as the event name and are not included in COMMON. Further, the statements for entering the event subroutines are unchanged from one simulation to another as contrasted to Deck MAIN of Method 1 which must be modified whenever an event is added or deleted. However, one special variable MYSELF is located in COMMON. Its use will be developed later.

Referring to the above example, assume that it is desirable to have event MOVE cause event DELAY T units of time in the future. Subroutine MOVE will contain the two following statements:

```
Subroutine MOVE

EXTERNAL DELAY
   .
    .
     .
CALL CAUSE (DELAY, TIME + T, 0, 0, 0)
   .
    .
     .
RETURN

END
```

032

When subroutine CAUSE is entered the address associated with the parameter DELAY is the address of the entry point in subroutine DELAY. Therefore, what gets stored in column 3 of the calender is the first executable instruction in subroutine DELAY. Thus, the event references mentioned above are the first instructions of the event subroutines. As will be apparent below, this Method 2 mechanism works because the first instruction of a subroutine is always a transfer to the prolog of the subroutine.

In deck MAIN, the subroutine selection statements are:

```
      1000  MYSELF  =  NEVENT (ITL)
            CALL TSTEP (MYSELF)
            GO TO 1000
```

When statement 1000 has been executed MYSELF contains the first instruction of the event routine to be entered. Following that, subroutine TSTEP is called with the address of MYSELF as the parameter address.

The form of TSTEP is:

```
      SUBROUTINE TSTEP (DUMMY)
      .
       .
        .
      IPAR 1 =
      IPAR 2 =
      IPAR 3 =

      CALL DUMMY (IPAR 1, IPAR 2, IPAR 3)

      RETURN

      END
```

The address of DUMMY is, remember, the address of MYSELF. The CALL DUMMY is translated into the following instructions:

```
      TSX   MYSELF, 4

      TXI      3

      PZE

      PZE   IPAR 1

      PZE   IPAR 2

      PZE   IPAR 3
```

033

The TSX MYSELF, 4 instruction causes the control to transfer to a location in COMMON with linkage established in index register 4. As mentioned above the first instruction of a FORTRAN IV subroutine compiled by IBSYS is always of the form:

```
          TRA  Prolog
```

Therefore, after the TSX transfers control to the location of MYSELF, the value of MYSELF transfers contol to the prologs of the desired event without modifying the return or parameter linkage. This is precisely the desired transfer.

The variable MYSELF serves one other important function. Because FORTRAN does not allow a routine to contain an EXTERNAL statement which contains the name of that routine, event routine MOVE cannot contain a statement of the form:

```
          CALL CAUSE (MOVE, . . . .).
```

However, the desired effect will be obtained using:

```
          CALL CAUSE (MYSELF, . . . ).
```

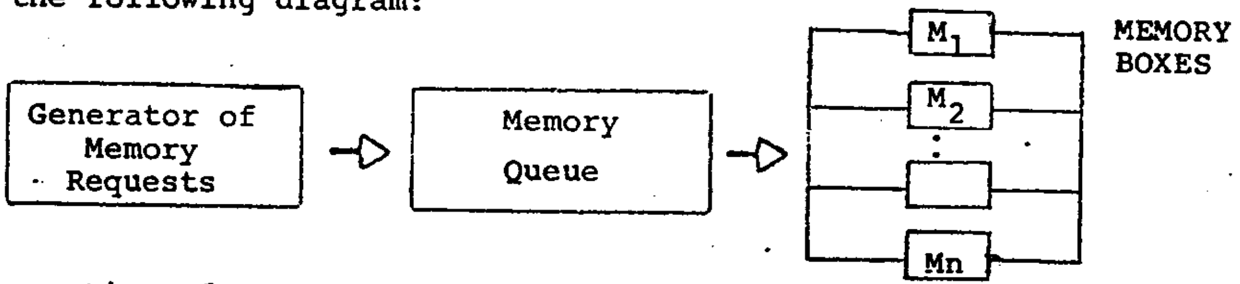
The complete listings of the utilities routines required for both Method 1 and Method 2 are given in the appendices.

EXAMPLE

An example will now be described which illustrates the details of implementation of a system simulation in FORTRAN IV.

The system under study will be a simple computer memory queue. Suppose a computer has several independent memory boxes. We may thus queue up memory requests and each computer cycle examine the queue and the memory boxes to see if there is a request on the queue for some non-busy box. A simulation will enable us to experimentally determine such things as average time on queue, average queue length, etc., as a function of request generation rate, number of memory boxes, and the memory cycle time.

The system may be roughly described as consisting of three parts, as in the following diagram:



The generation of memory requests will be artificially modeled by forming either no request or one request per computer cycle, according to some probability, with the box number of the request chosen at random. A generated request will be placed on the queue, if there is space for it. Every cycle, the queue will be scanned for the first request for a free memory box. If one is found it will then cause the memory box to be set busy for the cycle time.

A detailed description of the simulation now follows, with the FORTRAN IV event subroutines separately listed and described.

GENER

The simulation of the generation of requests is performed by GENER, a routine which first causes itself one cycle later and thus runs every cycle. GENER causes a request to be generated if a random number, uniformly distributed between 0 and 1, is found to be less than the specified probability of generating one request in a cycle. If the request is to be generated, a random number is then used to select a memory box for the request. If there is room on the queue, the request is caused to arrive at the queue.8 units of time later, at the "end" of the machine cycle. The listing of GENER follows.

035

```
$IBFTC GENER
      SUBROUTINE XGENER
      COMMON _ _ _
C     GENERATE MEMORY REQUEST
      CALL CAUSE (GENER, TIME + 1.0)
      CALL RANDOM (R)
      IF(R.GT. PROB1) RETURN
      CALL RANDOM (R)
      BOXNO = (R * FLOAT (NBOX)) + 1.0
      IF(QMPNT.EQ.NQM) RETURN
      INUMB = INUMB + 1
      CALL CAUSE (QBUSY, TIME + .8, BOXNO, INUMB)
      RETURN
      END
```

<u>QBUSY</u>

The event routine QBUSY simulates the arrival of a request on the queue. This is done by incrementing the queue input pointer QMPNT, and placing the instruction number and memory box number into the queue array QM.

```
$IBFTC QBUSY
      SUBROUTINE XQBUSY (BOXNO, INSTR)
      COMMON _ _ _
C     PLACE REQUEST ON QUEUE
      QMPNT = QMPNT + 1
      QM (QMPNT, 1) = INSTR
      QM (QMPNT, 2) = BOXNO
      RETURN
      END
```

<u>QMCON</u>

The simulation of the control of the queue is performed by QMCON. This event first causes itself to run again one cycle later. Then a scan pointer QMSCAN is initialized to one. The queue entry indicated by QMSCAN is then examined to see

636

if the indicated memory box is busy. If it is, the scan pointer is advanced and the next entry similarly examined. If the box is not busy, the memory request is issued by causing the events MBUSY and QUEMP at .8 units of time later (at the "end" of the cycle), and by causing the event MCYCC at a time .8 + CYCT later.

```
$IBFTC QMCON

      SUBROUTINE XQMCON

      COMMON _ _ _

C     QUEUE CONTROL, SCAN QUEUE AND

C     SEND OUT MEMORY REQUEST, IF POSSIBLE

      CALL CAUSE (QMCON, TIME + 1.0)

      QMSCAN = 1

10    IF(QMSCAN, GT, QMPNT) RETURN

      BOXNO = QM(QMSCAN, 2)

      INSTR = QM(QMSCAN, 1)

      IF(MEMBSY (BOXNO).EQ. 1) GO TO 20

      CALL CAUSE (MBUSY, TIME + .8, BOXNO, INSTR)

      CALL CAUSE (QUEMP, TIME + .8, QMSCAN)

      CALL CAUSE (MCYCC, TIME + .8 + CYCT, BOXNO)

      RETURN

20    QMSCAN = QMSCAN + 1

      IF(QMSCAN.GT.NQM) RETURN

      GO TO 10

      END
```

<u>MBUSY</u>

This event sets the indicated memory box busy by placing INSTR into position BOXNO the array MEMBSY.

```
$IBFTC MBUSY

      SUBROUTINE XMBUSY(BOXNO, INSTR)

      COMMON _ _ _
```

037

```
      C   PLACE REQUEST INSTR IN MEMORY BOXNO
          MEMBSY (BOXNO) = INSTR
          RETURN
          END
```

QUEMP

This event removes the indicated entry from the queue, "moves up" any following entries, and decrements the input pointer.

```
  $IBFTC QUEMP
          SUBROUTINE XQUEMP (QMSCAN)
          COMMON _ _ _
      C   REMOVE REQUEST AT QMSCAN FROM QUEUE
          J = NQM - L
          DO 9 L = 1, 10
          DO 7 K = QMSCAN, J
      7   QM(K,L) = QM(K + 1, L)
      9   QM (NQM,L) = 0
          QMPNT = QMPNT - 1
          RETURN
          END
```

MCYCC

This event simulates the completion of the memory cycle by resetting the memory busy indicator of the specified memory box.

```
  $ IBFTC MCYCC
          SUBROUTINE XMCYCC (BOXNO)
          COMMON _ _ _
      C   AT MEMORY CYCLE COMPLETION, FREE BOX
          MEMBSY (BOXNO) = 0
          RETURN
          END
```

038

STATS

Included in the list of events is one called STATS which is an output routine. STATS causes itself one cycle later, and outputsthe current system status. The run stops if a specified value of simulated time MAXT is exceeded.

```
     $IBFTC STATS

            SUBROUTINE XSTATS

            COMMON _ _ _

       C    STATS IS THE OUTPUT ROUTINE

            CALL CAUSE (STATS, TIME + 1.0)
             / / / / / / / / / / / /
            / / / / / / / / / / / /
            COLLECT AND OUTPUT SYSTEM STATUS
            / / / / / / / / / / / /
           / / / / / / / / / / / /
            IF(TIME .GE.MAXT) STOP

            RETURN

            END
```

RANDOM

Random is a random number generator. The statement CALL RANDOM(R) returns R to the calling routine a value between 0 and 1 with uniform distribution.

CAUSE

CAUSE is one of the simulation utility subroutines previously specified in this report. It is called to place an event into the calender.

TSTEP

TSTEP is one of the simulation utility subroutines previously specified in this report. It is called from MAIN to advance simulated time to that of the next event in time, and get the parameters and number of that event.

MAIN

MAIN is the first entered and "main" routine of the simulation program and performs a number of functions. First it initializes the common variables to zero. Then the run parameters are read into the appropriate common variables. The calender is then initialized with the proper linkage and starting events are placed into the calender with CAUSE statements. Following and

039

including the statement number 1000 in MAIN are the instruction necessary to cycle thru the events in the calender.

Assume that the following COMMON and specification statements are included in every routine described, and indicated by the statement: COMMON _ _ _

```
      COMMON TIME, IPAR 1, IPAR 2, IPAR 3, ID, ISL, ITL,
     1 LINK (200),CTIME (200), NEVENT (200), KOLI (200),
     2 KOL2(200), KOL3(200), NBOX, NQM, CYCT, MAXT,
     3 PROB1, QM (32,2), MEMBSY(64), QMPNT, INUMB
      INTEGER QM,QMPNT
      REAL    MAXT
      COMMON / NAMES / GENER, QBUSY, QMCON, MBUSY
     1 QUEMP, MCYCC, STATS
      INTEGER GENER, QBUSY, QMCON, QUEMP, STATS
```

The listing of MAIN follows.

```
$IBFTC MAIN
      COMMON _ _ _
      EQUIVALENCE (COM(1), TIME), (X, CTIMEC(1))
C     MAIN INITIALIZES COMMON TO ZEROES, READS IN
C     SYSTEM PARAMETERS, SETS UP THE CALENDER, INITIALIZES
C     THE EVENT VALUES, PLACES STARTING EVENTS INTO THE
C     CALENDER AND THEN CONTROLS THE SEQUENCING OF EVENTS
      DO 101 I = 1,3000
  101 COM (I) = 0
      READ  PROB1, CYCT, NQM, NBOX, MAXT
      TIME = 0.0
      DO 92 ITL = 2,199
   92 LINK (ITL) = ITL + 1
      ISL = 2
      ITL = 1
      X = 1.0E30
      GENER = 1
```

040

```
          QBUSY = 2
          QMCON = 3
          MBUSY = 4
          QUEMP = 5
          MCYCC = 6
          STATS = 7
          CALL CAUSE (STATS, TIME + 1.0)
          CALL CAUSE (QMCON, TIME + 1.1)
          CALL CAUSE (GENER, TIME + 1.1)
   1000   CALL TSTEP (EVENT)
          GO TO (1, 2, 3, 4, 5, 6, 7), EVENT
      1   CALL GENER
          GO TO 1000
      2   CALL XQBUSY (IPAR 1, IPAR 2)
          GO TO 1000
      3   CALL XQMCON
          GO TO 1000
      4   CALL XMBUSY (IPAR 1, IPAR 2)
          GO TO 1000
      5   CALL XQUEMP (IPAR 1)
          GO TO 1000
      6   CALL XMCYCC (IPAR 1)
          GO TO 1000
      7   CALL XSTATS
          GO TO 1000
          END
```

041

REFERENCES

1. K. Blake and G. Gordon, "Systems Simulation with Digital Computers," IBM Systems Journal, Vol. 3, No. 1, 14 (1964).

2. R. Efron and G. Gordon, "A General Purpose Digital Simulator and Examples of its Application: Part I Description of the Simulator, " IBM Systems Journal, Vol. 3, No. 1, 22 (1964).

3. "General Purpose Systems Simulator II," Form B20-6346-1, International Business Machines Corporation.

4. B. Dimsdale and H. M. Markowitz, "A Description of the SIMSCRIPT Language," IBM Systems Journal, Vol. 3, No. 1, 57 (1964).

5. H. M. Markowitz, B. Hausner, and H. W. Karr, "SIMSCRIPT, A Simulation Programming Language," The RAND Corporation, 1963, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Appendix A

Listings of utility routines for Method 1

043

```
      SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C                CAUSE ENTERS EVENTS ONTO CALENDAR
C                ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C                ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      NEXT=ITL
      GO TO 20
C  10            LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
   10 LAST=NEXT
      NEXT=LINK(NEXT)
   20 IF (T .GT. CTIME(NEXT)) GO TO 10
      ID=ISL
      ISL=LINK(ISL)
      LINK(ID)=NEXT
C                SEE IF THIS EVENT WILL BE THE FIRST ON THE LIST
      IF (NEXT.EQ. ITL)GO TO 40
      LINK(LAST)=ID
   30 CTIME (ID)=T
      NEVENT(ID)=IEV
      KOL1(ID)=IP1
      KOL2(ID)=IP2
      KOL3(ID)=IP3
      RETURN
   40 ITL=ID
      GO TO 30
      END
```

044

```
      SUBROUTINE REMOVE(EVENT,STIME,I,J,K)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
      INTEGER EVENT
      NEXT=ITL
      IF(NEVENT(ITL).EQ.EVENT) GO TO 20
   10 LAST=NEXT
      NEXT=LINK(NEXT)
      IF(NEXT.EQ.0) GO TO 30
      IF(NEVENT(NEXT).NE.EVENT) GO TO 10
C              WE FOUND EVENT
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      LINK(LAST)=LINK(NEXT)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C              EVENT IS FIRST IN LIST
   20 CONTINUE
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      ITL=LINK(ITL)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C              EVENT NOT PRESENT
   30 CONTINUE
      STIME=TIME
      I=0
      J=0
      K=0
      RETURN
      END
```

045

```
      SUBROUTINE TSTEP(IEVENT)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C                SUBROUTINE TO STEP EVENTS IN CALENDAR
C                ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C                ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      ID=ITL
      ITL=LINK(ID)
      LINK(ID)=ISL
      ISL=ID
      TIME=CTIME(ID)
      IPAR1=KOL1(ID)
      IPAR2=KOL2(ID)
      IPAR3=KOL3(ID)
      IEVENT=NEVENT(ID)
      RETURN
      END
```

046

Appendix B

Listings of utility routines for Method 2

047

```
      SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C               CAUSE ENTERS EVENTS ONTO CALENDAR
C               ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C               ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      NEXT=ITL
      GO TO 20
C  10           LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
   10 LAST=NEXT
      NEXT=LINK(NEXT)
   20 IF (T .GT. CTIME(NEXT)) GO TO 10
      ID=ISL
      ISL=LINK(ISL)
      LINK(ID)=NEXT
C               SEE IF THIS EVENT WILL BE THE FIRST ON THE LIST
      IF (NEXT.EQ. ITL)GO TO 40
      LINK(LAST)=ID
   30 CTIME (ID)=T
      NEVENT(ID)=IEV
      KOL1(ID)=IP1
      KOL2(ID)=IP2
      KOL3(ID)=IP3
      RETURN
   40 ITL=ID
      GO TO 30
      END
```

048

```
      SUBROUTINE REMOVE(EVENT,STIME,I,J,K)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
      INTEGER EVENT
      NEXT=ITL
      IF(NEVENT(ITL).EQ.EVENT) GO TO 20
   10 LAST=NEXT
      NEXT=LINK(NEXT)
      IF(NEXT.EQ.0) GO TO 30
      IF(NEVENT(NEXT).NE.EVENT) GO TO 10
C                WE FOUND EVENT
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      LINK(LAST)=LINK(NEXT)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C                EVENT IS FIRST IN LIST
   20 CONTINUE
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      ITL=LINK(ITL)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C                EVENT NOT PRESENT
   30 CONTINUE
      STIME=TIME
      I=0
      J=0
      K=0
      RETURN
      END
```

049

```
      SUBROUTINE TSTEP(DUMMY)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C                SUBROUTINE TO STEP EVENTS IN CALENDAR
C                ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C                ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      ID=ITL
      ITL=LINK(ID)
      LINK(ID)=ISL
      ISL=ID
      TIME=CTIME(ID)
      IPAR1=KOL1(ID)
      IPAR2=KOL2(ID)
      IPAR3=KOL3(ID)
      CALL DUMMY(IPAR1,IPAR2,IPAR3)
      RETURN
      END
```

050