GASPL/I: A PL/I BASED SIMULATION LANGUAGE

Robert E. Young and

A. Alan B. Pritsker

Purdue University

ABSTRACT

GASPL/I is a PL/I based simulation language structured around the GASP simulation philosophy. It is a discrete-event simulator which extends and refines the principles employed in the FORTRAN based GASP II language. In this paper, an example of the use of GASPL/I is presented. The example involves the simulation of a project when resources are limited. Emphasis is given to the file system and the multiple-entry feature of GASPL/I. A portion of the GASPL/I coding for the example is included.

Introduction

The purpose of this paper is to demonstrate the file system capabilities of GASPL/I. The presentation will be in the context of an example problem in lieu of a detailed explanation of the language. The problem statement for the example will be given first. The procedure for solving the problem using GASPL/I is subsequently detailed. The coding for the example is presented along with a list of the PL/I subprograms which comprise GASPL/I. Comments are made on GASPL/I capabilities and developments.*

Problem Statement

A simulation program is to be developed to evaluate the effects of dispatching rules for assigning personnel in a development project. The project is modeled as a network. Four dispatching rules have been suggested for deciding the order in which the activities of the project should be performed. Evaluation of the dispatching rules is based on project completion time with low values preferred. The network is shown in Figure 1. Table 2 lists the resources required to perform each activity and the characteristics of the time to perform each activity. As seen from the table, the time to perform an activity may be a random variable. The resources available to the project are listed in Table 1.

Table 1. Resource Availability

Type	Description	Availability
1	Systems Analyst	4
2	Marketing Personnel	3
3	Maintenance Personnel	5
4	Engineering Personnel	4

The decision rules to be tested are as follows:

1. Dispatch an activity with the lowest requirement for Systems Analyst first.
2. Dispatch an activity with the lowest requirement for Marketing Personnel first.
3. Dispatch an activity with the lowest requirement for Maintenance Personnel first.
4. Dispatch an activity with the lowest requirement for Engineering Personnel first.

If a dispatching rule results in a tie, the activity that has been waiting the longest for dispatching will be chosen.

For each resource type a list of the nodes and their associated activities ranked according to decreasing resource requirements is to be printed. The network is to be simulated 100 times for each decision rule.

* The authors are indebted to Professor L. P. McNamee of the UCLA School of Engineering and Applied Science for his help in testing the support programs.

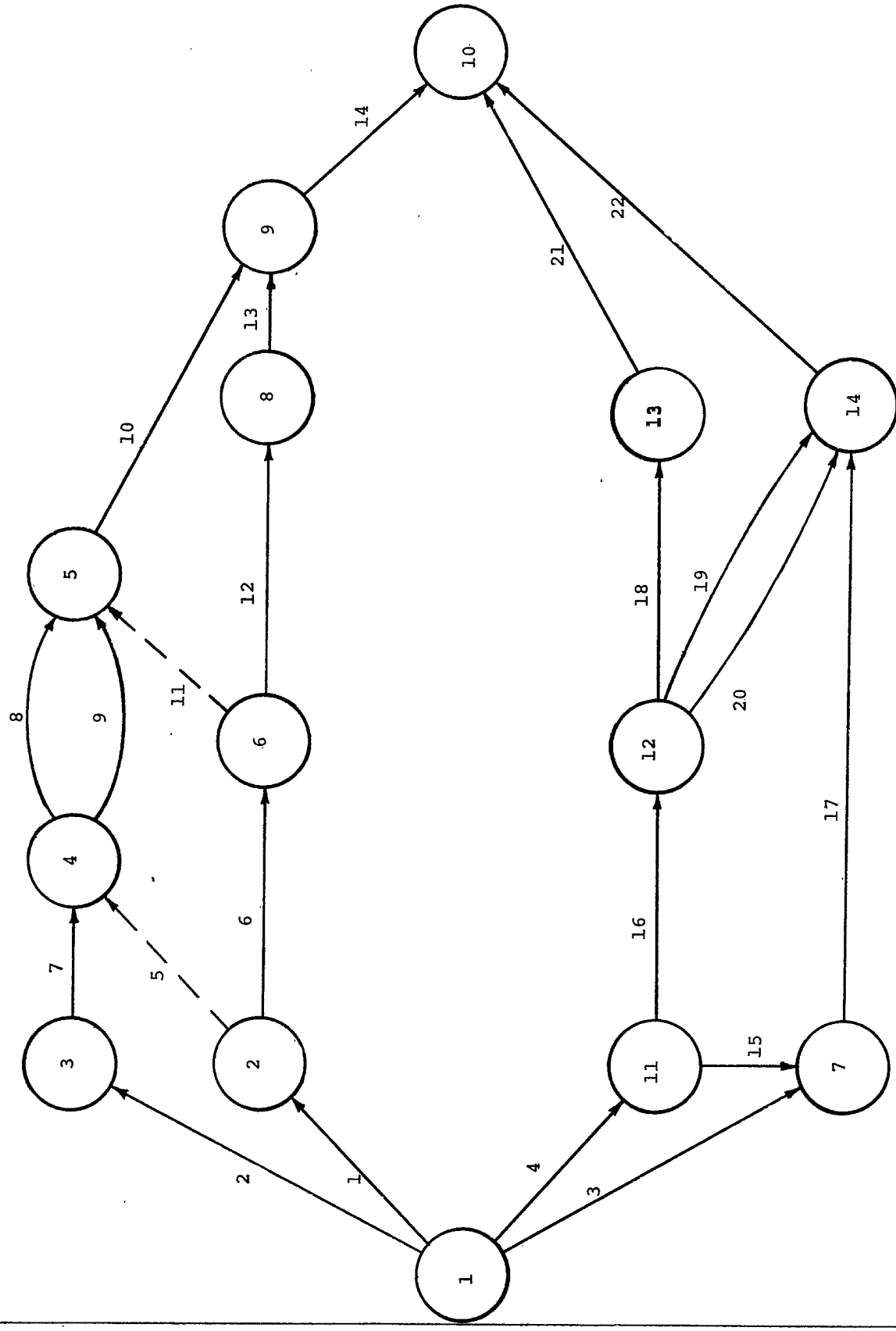


Figure 1
Project Network

TABLE 2

Network Parameters

Activity Number	Distribution	Mean Time	Variance	Requirement for Resource Type 1	2	3	4
1	Lognormal	7	2	2	0	1	3
2	Constant	3	0	2	0	1	1
3	Constant	10	0	0	2	0	0
4	Constant	5	0	1	1	1	1
5	Dummy	0	0	0	0	0	0
6	Constant	5	0	2	0	3	2
7	Uniform	3	1/3	1	0	2	3
8	Gamma	8	6	3	0	0	1
9	Constant	3	0	1	0	0	0
10	Constant	5	0	2	0	1	3
11	Dummy	0	0	0	0	0	0
12	Constant	2	0	2	1	2	2
13	Constant	6	0	1	1	0	0
14	Constant	10	0	0	0	0	0
15	Exponential	10	0	1	1	1	0
16	Constant	3	0	1	1	2	0
17	Constant	5	0	0	1	1	1
18	Constant	2	0	0	1	0	3
19	Normal	5	4	1	0	0	2
20	Constant	1	0	1	1	0	0
21	Constant	15	0	0	1	2	0
22	Constant	5	0	0	1	0	0

Procedure

The philosophy for simulating networks is discussed elsewhere [1,2]. Only one event routine is required. This event will be coded as the procedure EVNTS, and EVNTS performs all operations required when an end-of-activity event occurs. When a simulation of the network is completed, procedure ENDNET initializes the variables to start a new simulation and when the specified 100 passes through the network for a given decision rule have been completed, ENDNET establishes a new decision rule. Figure 2 is a listing of procedure EVNTS and Figure 3 is a listing of procedure ENDNET.

When an end-of-activity event occurs, the resources used to perform the completed activity are made available for use in dispatching. If the end node of the activity is realized, the activities emanating from that node are available for dispatching. The dispatching rule is used to assign available resources to the activities that can be started.

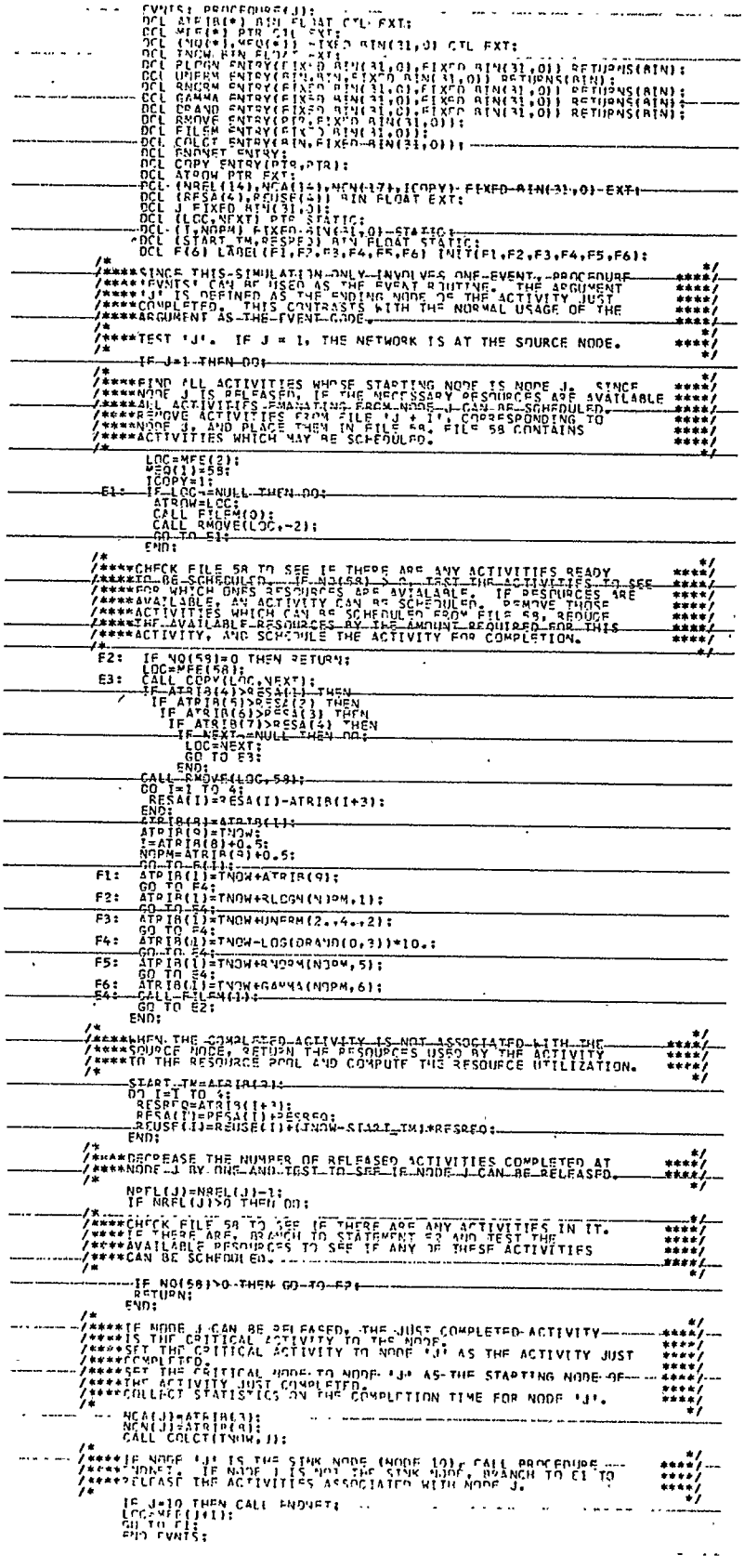
The activities of the network are the entities of the system. These entities must be grouped by their start node so that they can be made available when their start node is realized.

Reports are to be generated for each resource, listing the activities emanating from a node ranked according to resource requirement. This suggests the necessity of writing a sorting routine. This is not the case since the activities associated with each node can be sorted using the multiple entry feature of GASPL/I. To accomplish the required ranking, each entity is stored only once, but is placed in four separate files. Each of these four files is ranked on a different resource type, where a given resource corresponds to a different attribute of the entity. In this way, the activities associated with a node are physically stored only once, but belong to four files. When the reports are to be printed, the appropriate files are accessed.

The project involves 14 nodes and therefore 56 (14*4) files will be used to store the activities emanating from each node. The definitions of the attributes associated with each file are shown in Table 3. File 1 is used to store all events. The 22 activities are stored in files 2 through 57 and file 58 contains the activities that can be dispatched; i.e., those activities whose predecessors have been completed. Files 2 through 15 are ranked by attribute four, corresponding to Systems Analysts. File 16 through 29 are ranked by attribute five, corresponding to Marketing Personnel. Files 30 through 43 are ranked by attribute six, Maintenance Personnel. Files 44 through 57 are ranked by attribute seven, Engineering Personnel.

Activities emanating from node 1 are stored in files 2, 16, 30 and 44; from node 2 in files 3, 17, 31 and 45; etc. Thus, activities emanating from node j are stored in files j+1+ (i*14), i=0,1,2, 3.

Although there are four files for each



```
EVNTS: PROCEDURE(J);
DCL ATRIB(*) BIN FLOAT CTL EXT;
DCL MFE(*) PTR CTL EXT;
DCL (NQ(*),MFQ(*)) FIXED BIN(31,0) CTL EXT;
DCL TNOW BIN FLOAT EXT;
DCL PLOGN ENTRY(FIXED BIN(31,0),FIXED BIN(31,0)) RETURNS(BIN);
DCL UNIFRM ENTRY(BIN,BIN,FIXED BIN(31,0)) RETURNS(BIN);
DCL RNORM ENTRY(FIXED BIN(31,0),FIXED BIN(31,0)) RETURNS(BIN);
DCL GAMMA ENTRY(FIXED BIN(31,0),FIXED BIN(31,0)) RETURNS(BIN);
DCL DRAND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0)) RETURNS(BIN);
DCL RMOVE ENTRY(PTR,FIXED BIN(31,0));
DCL FILEM ENTRY(PTR) BIN(31,0);
DCL COLCT ENTRY(BIN,FIXED BIN(31,0));
DCL ENDNET ENTRY;
DCL COPY ENTRY(PTR,PTR);
DCL ATROW PTR EXT;
DCL (NREL(14),NCA(14),NCN(17),ICOPY) FIXED BIN(31,0) EXT;
DCL (RESA(4),RESE(4)) BIN FLOAT EXT;
DCL J FIXED BIN(31,0);
DCL (LOC,NEXT) PTR STATIC;
DCL (I,NOPM) FIXED BIN(31,0) STATIC;
DCL (START_TM,RESPE) BIN FLOAT STATIC;
DCL F(6) LABEL(F1,F2,F3,F4,F5,F6) INIT(F1,F2,F3,F4,F5,F6);
/*
/****SINCE THIS SIMULATION ONLY INVOLVES ONE EVENT, PROCEDURE ****/
/****'EVNTS' CAN BE USED AS THE EVENT ROUTINE. THE ARGUMENT ****/
/****'J' IS DEFINED AS THE ENDING NODE OF THE ACTIVITY JUST ****/
/****COMPLETED. THIS CONTRASTS WITH THE NORMAL USAGE OF THE ****/
/****ARGUMENT AS THE EVENT CODE. ****/
/* */
/****TEST 'J'. IF J = 1, THE NETWORK IS AT THE SOURCE NODE. ****/
/* */
IF J=1 THEN DO;
/* */
/****FIND ALL ACTIVITIES WHOSE STARTING NODE IS NODE J. SINCE ****/
/****NODE J IS RELEASED, IF THE NECESSARY RESOURCES ARE AVAILABLE ****/
/****ALL ACTIVITIES EMANATING FROM NODE J CAN BE SCHEDULED. ****/
/****REMOVE ACTIVITIES FROM FILE 'J + 1', CORRESPONDING TO ****/
/****NODE J, AND PLACE THEM IN FILE 58. FILE 58 CONTAINS ****/
/****ACTIVITIES WHICH MAY BE SCHEDULED. ****/
/* */
LOC=MFE(2);
MFE(1)=58;
ICOPY=1;
E1: IF LOC^=NULL THEN DO;
ATROW=LOC;
CALL FILEM(0);
CALL RMOVE(LOC,-2);
GO TO E1;
END;
/*
/****CHECK FILE 58 TO SEE IF THERE ARE ANY ACTIVITIES READY ****/
/****TO BE SCHEDULED. IF NQ(58) > 0, TEST THE ACTIVITIES TO SEE ****/
/****FOR WHICH ONES RESOURCES ARE AVAILABLE. IF RESOURCES ARE ****/
/****AVAILABLE, AN ACTIVITY CAN BE SCHEDULED. REMOVE THOSE ****/
/****ACTIVITIES WHICH CAN BE SCHEDULED FROM FILE 58, REDUCE ****/
/****THE AVAILABLE RESOURCES BY THE AMOUNT REQUIRED FOR THIS ****/
/****ACTIVITY, AND SCHEDULE THE ACTIVITY FOR COMPLETION. ****/
/* */
E2: IF NQ(58)=0 THEN RETURN;
LOC=MFE(58);
E3: CALL COPY(LOC,NEXT);
IF ATRIB(4)>RESA(1) THEN
GO TO E4;
IF ATRIB(5)>RESA(2) THEN
GO TO E4;
IF ATRIB(6)>RESA(3) THEN
GO TO E4;
IF ATRIB(7)>RESA(4) THEN
GO TO E4;
IF NEXT^=NULL THEN DO;
LOC=NEXT;
GO TO E3;
END;
CALL RMOVE(LOC,58);
DO I=1 TO 4;
RESA(I)=RESA(I)-ATRIB(I+3);
END;
ATRIB(8)=ATRIB(1);
ATRIB(9)=TNOW;
I=ATRIB(8)+0.5;
NOPM=ATRIB(9)+0.5;
GO TO F(I);
F1: ATRIB(1)=TNOW+ATRIB(9);
GO TO E4;
F2: ATRIB(1)=TNOW+RLOGN(NOPM,1);
GO TO E4;
F3: ATRIB(1)=TNOW+UNFRM(2.,4.,2);
GO TO E4;
F4: ATRIB(1)=TNOW-LOG(DRAND(0,3))*10.;
GO TO E4;
F5: ATRIB(1)=TNOW+RNORM(NOPM,5);
GO TO E4;
F6: ATRIB(1)=TNOW+GAMMA(NOPM,6);
E4: CALL FILEM(1);
GO TO E2;
END;
/*
/****WHEN THE COMPLETED ACTIVITY IS NOT ASSOCIATED WITH THE ****/
/****SOURCE NODE, RETURN THE RESOURCES USED BY THE ACTIVITY ****/
/****TO THE RESOURCE POOL AND COMPUTE THE RESOURCE UTILIZATION. ****/
/* */
START_TM=ATRIB(2);
DO I=1 TO 4;
RESREQ=ATRIB(I+3);
RESA(I)=RESA(I)+RESREQ;
REUSE(I)=REUSE(I)+(TNOW-START_TM)*RESREQ;
END;
/*
/****DECREASE THE NUMBER OF RELEASED ACTIVITIES COMPLETED AT ****/
/****NODE J BY ONE AND TEST TO SEE IF NODE J CAN BE RELEASED. ****/
/* */
NREL(J)=NREL(J)-1;
IF NREL(J)>0 THEN DO;
/* */
/****CHECK FILE 58 TO SEE IF THERE ARE ANY ACTIVITIES IN IT. ****/
/****IF THERE ARE, BRANCH TO STATEMENT E2 AND TEST THE ****/
/****AVAILABLE RESOURCES TO SEE IF ANY OF THESE ACTIVITIES ****/
/****CAN BE SCHEDULED. ****/
/* */
IF NQ(58)>0 THEN GO TO E2;
RETURN;
END;
/*
/****IF NODE J CAN BE RELEASED, THE JUST COMPLETED ACTIVITY ****/
/****IS THE CRITICAL ACTIVITY TO THE NODE. ****/
/****SET THE CRITICAL ACTIVITY TO NODE 'J' AS THE ACTIVITY JUST ****/
/****COMPLETED. ****/
/****SET THE CRITICAL NODE TO NODE 'J' AS THE STARTING NODE OF ****/
/****THE ACTIVITY JUST COMPLETED. ****/
/****COLLECT STATISTICS ON THE COMPLETION TIME FOR NODE 'J'. ****/
/* */
NCA(J)=ATRIB(3);
NCN(J)=ATRIB(8);
CALL COLCT(TNOW,J);
/*
/****IF NODE 'J' IS THE SINK NODE (NODE 10), CALL PROCEDURE ****/
/****'ENDNET'. IF NODE J IS NOT THE SINK NODE, BRANCH TO E1 TO ****/
/****RELEASE THE ACTIVITIES ASSOCIATED WITH NODE J. ****/
/* */
IF J=10 THEN CALL ENDNET;
LOC=MFE(J+1);
GO TO E1;
END EVNTS;
```

Figure 2

PL/I Listing of Procedure EVNTS

```
      ENDNET: PROCEDURE;
      DCL (ATRIB(*),ENQ(*),VNQ(*),QTIME(*),SUMA(*,*)) BIN FLOAT CTL EXT;
      DCL (JCELS(*),KRANK(*),MAXNQ(*)) FIXED BIN(31,0) CTL EXT;
      DCL (NCNT,NREL(14),NRELP(14),NRUN,NRUNS,NUMC(22),MSTOP,
           NA(16),NCA(14),NCN(14)) FIXED BIN(31,0) EXT;
      DCL (RESUT(4),REUSE(4),RESAP(4),TNOW,TBEG,RESA(4)) BIN FLOAT EXT;
      DCL COLCT ENTRY(BIN,FIXED BIN(31,0));
      DCL HISTO ENTRY(BIN,FIXED BIN(31,0));
      DCL SUMRY ENTRY;
      DCL (LCN,I,LCA) FIXED BIN(31,0) STATIC;
      DCL XNUM BIN FLOAT STATIC;
/*                                                                    */
/****COLLECT STATISTICS ON THE COMPLETION TIME AND                    ****/
/****THE RESOURCE UTILIZATION.                                        ****/
/*                                                                    */
      CALL HISTO(TNOW,1);
      DO I=1 TO 4;
       RESUT(I)=REUSE(I)/(TNOW*RESAP(I));
      CALL COLCT(RESUT(I),14+I);
      END;
/*                                                                    */
/****TRACE BACK THROUGH THE NETWORK TO DETERMINE THE CRITICAL         ****/
/****PATH; THEN DECREASE THE NUMBER OF NETWORK SIMULATIONS            ****/
/****REMAINING, NCNT, BY ONE; AND TEST TO SEE IF THE DESIRED          ****/
/****NUMBER OF SIMULATIONS FOR THIS RUN HAS BEEN COMPLETED.           ****/
/*                                                                    */
      LCN=10;
      LCA=NCA(10);
   N1: NUMC(LCA)=NUMC(LCA)+1;
      LCN=NCN(LCN);
      IF LCN¬=1 THEN DO;
       LCA=NCA(LCN);
       GO TO N1;
      END;
      NCNT=NCNT-1;
      IF NCNT=0 THEN DO;
/*                                                                    */
/****IF THE DESIRED NUMBER OF SIMULATIONS FOR THIS RUN HAVE BEEN      ****/
/****COMPLETED, COMPUTE AND PRINT THE CRITICALITY INDEX,              ****/
/****CALL SUMRY, AND INITIALIZE FOR THE NEXT SIMULATION RUN.          ****/
/*                                                                    */
       PUT EDIT('*** CRITICAL PATH INDICES ***')(PAGE,SKIP(2),X(15),A);
       PUT EDIT('DISPATCHING RULE NO.',NRUN)(SKIP(2),X(16),A,F(2));
       PUT SKIP;
       DO I=1 TO 22;
        XNUM=FLOAT(NUMC(I))/SUMA(10,3);
        PUT EDIT('CRITICALITY INDEX FOR ACTIVITY',I,'IS',XNUM)
             (SKIP(2),X(5),A,F(3),A,F(10,4));
       END;
       CALL SUMRY;
/*                                                                    */
/****IF NRUNS > 1, NOT ALL THE DECISION RULES HAVE BEEN TESTED.       ****/
/****DECREASE NRUNS BY ONE, INCREASE NRUN BY ONE, AND ADD ONE         ****/
/****TO KRANK(58).  BY ADDING ONE TO KRANK(58), THE RANKING           ****/
/****OF ENTRIES IN FILE 58 FOR SUBSEQUENT RUNS WILL BE BASED          ****/
/****UPON THE NEXT ATTRIBUTE.  IN THIS WAY, A NEW DECISION            ****/
/****RULE CAN BE IMPLEMENTED.  INITIALLY FILE 58 IS RANKED            ****/
/****BY ATTRIBUTE FOUR (I.E., KRANK(58) IS SET TO 4 BY THE            ****/
/****INPUT DATA).                                                     ****/
/****RESET THE STATISTICS AND NECESSARY VARIABLES FOR TESTING         ****/
/****ANOTHER DECISION RULE.                                           ****/
/****                                                                 ****/
/****IF NRUNS = 1, ALL THE DECISION RULES HAVE BEEN TESTED.           ****/
/****ASSIGN VALUES TO THE NECESSARY GASP/I VARIABLES TO               ****/
/****TERMINATE THE SIMULATION AND RETURN.                             ****/
/*                                                                    */
       IF NRUNS>1 THEN DO;
        KRANK(58)=KRANK(58)+1;
        NRUNS=NRUNS-1;
        NRUN=NRUN+1;
       END;
       ELSE DO;
        NA(13)=1;
        MSTOP=-1;
       RETURN;
       END;
       SUMA=0.;
       SUMA(*,4)=1.0E20;
       SUMA(*,5)=-1.0E20;
       JCELS=0;
       NCNT=100;
       NUMC=0;
      END;
/*                                                                    */
/****IF THE NUMBER OF SIMULATIONS FOR THIS RUN HAVE NOT BEEN          ****/
/****COMPLETED, RESET THE FILE STATISTICS, SCHEDULE AN INITIAL        ****/
/****EVENT TO START THE NEXT SIMULATION, AND REINITIALIZE             ****/
/****THE NUMBER OF RELEASES FOR EACH NODE.                            ****/
/*                                                                    */
      TNOW,TBEG=0.;
      MAXNQ,ENQ,VNQ,QTIME=0.;
      ATRIB=0.;
      ATRIB(2)=1.;
      CALL FILEM(1);
      RSUSE,RESUT=0;
      RESA=RESA;
      NREL=NRELP;
      RETURN;
      END ENDNET;
```

Figure 3

PL/I Listing of Procedure ENDNET

node, only one file set will be used during the simulation; file set 2 through 15. When a node is realized, the activities emanating from that particular node are integrated into file 58. The storage of these entity sets is not duplicated but a pointer system is employed to link these entities with those already existing in file 58.

File 58 is ranked low-value-first using the dispatching rule being tested.

The above problem utilizes the GASPL/I filing system which is capable of entering a single entity into multiple files while storing the entity only once. In addition, once an entity is placed in a file, it can be entered into additional files without requiring duplicate storage for the attributes of the entity. The multiple entry feature allows the user increased program capability without a substantial increase in the required central memory or in execution efficiency.

Simulation Coding

A list giving the functional description of the GASPL/I routines is presented in Table 4. These routines are used to perform the time-advance function, the initialization, the file maintenance, the statistical collection and the input/output capabilities for a simulation program. The GASP/I user need only perform the tasks related to the coding of the specific problem being analyzed. It is this coding that is presented and discussed below.

Initialization of non-GASP variables is accomplished by the main program. Table 5 defines the non-GASP variables for the simulation of a project network. Procedure DATIN initializes the GASP variables, allocates storage for the various arrays, and loads desired information into the files. To begin the simulation, an end-of-activity event is scheduled for the source node (node 1) at time zero.

When node j is realized, the entities emanating from node j are entered into file 58 from file (j+1). The entries in file 58 are accessed sequentially by procedure COPY until an activity is located consistent with the available resources. The activity is scheduled for completion and its resources are subtracted from the resource pool. The process continues until insufficient resources remain to schedule further activities.

When an activity is completed, the resources used on the activity are returned to the resource pool and resource utilization is calculated. The end node of the activity is then tested for realization. If it is realized, statistics are collected and the previous procedure is repeated. If it is not realized, file 58 is accessed to dispatch activities using the available resources which now include the resources that were used on the activity just completed.

If the node realized is the sink node (node 10), procedure ENDNET is called to initialize the program for subsequent passes through the network and to print output information. When all passes through the network have been completed for each decision rule, this procedure sets the GASP variables to indicate the end of the simulation and a return is made to procedure GASP; and subsequently, to the main program to terminate execution. During the termination process, procedure GASP has the standard GASP summary report printed and calls procedure OUTPT to print special reports. A listing of Procedure OUTPT is in Figure 4.

General Comments

GASPL/I has all the capabilities of GASP II and makes the GASP philosophy and language available to PL/I programmers. In addition, GASPL/I has the following features:

1. Dynamic storage allocation.
2. Extended and simplified input/output capabilities.
3. Execution efficiency comparable to its FORTRAN counterpart.

The allocation of storage dynamically eliminates the overhead generated by carrying unnecessary core storage during execution.

The new input/output capabilities include unformated input with a multiple-file entry specification for entities, expanded simulation report output, and expanded statistical reporting. The depth and dimension of the output is easily tailored to the user needs without re-programming.

In examples tested to date, the GASPL/I language has demonstrated greater execution efficiency than the FORTRAN based GASP II [1] and the revised version, GASPU [3]. Currently, a PL/I version of GASP IV [4,5] is being developed.

TABLE 3

Definitions of the Attributes
Associated with each File

	1	2	3	4	5	6	7	8	9
File 1	End-of-activity time	End Node	Activity number	Type 1 Resources	Type 2 Resources	Type 3 Resources	Type 4 Resources	Start Node	Start time for an activity
Files 2 through 58	Start Node	End Node	Activity number	Type 1 Resources	Type 2 Resources	Type 3 Resources	Type 4 Resources	Distribution Type	Parameter Number containing information necessary to generate distribution or, if a constant, the constant time value

Files (j+1+(i*14), i = 0, 1, 2, 3) correspond to node j, j=1, ..., 14.

All files are ranked low-value first.

TABLE 4

Functional Description of GASPL/I Routines

Procedure	Description
GASP	Controling routine for the simulation.
DATIN	Performs data input and initialization.
SET	Initializes the filing system.
CLEAR	Clears the statistical arrays.
FILEM	Stores the vector ATRIB in a file.
RMOVE	Removes an entry from a file and places it's attributes in the vector ATRIB.
COPY	Duplicates an entry of the filing system into the vector ATRIB.
SUMRY	Prints the final GASPL/I summary report.
MONTR	Enables the user to monitor the simulation programs during user specified periods of the simulation.
PRNTQ	Prints the contents of a given file and associated statistics.
COLCT	Collects and prints statistics for variables based upon observation.
TIMST	Collects and prints statistics for time-persistent variables.
HISTO	Collects data for, and plots, histograms.
ERROR	Prints error messages and statistics to aid in debugging programs.
SUMQ	Computes the sum of all attribute values stored in a specified column of the specified file.
PRODQ	Computes the product of attribute values stored in a specified column of the specified file.
FIND	Locates a row in a specified file with a specified relationship to a given value.
DRAND	Random number generator, capable of generating multiple streams.
Random Deviate Generators	Generates deviates from β, γ, Poisson, normal, lognormal, Erlang, exponential, uniform distributions.

TABLE 5

Non-GASPL/I Variables

Non-GASPL/I Variables	Definition	Initial Value
NCA(J) J = 1,14	The activity which is critical to node J in a given simulation run.	0
NCN(J) J = 1,14	The node which is critical to node J in a given simulation run.	0
NCNT	The counter for the number of simulations for a given decision rule.	100
NUMC(K) K = 1,22	The number of times activity K is critical.	0
NREL(J) J = 1,14	The number of releases remaining for node J.	(0,1,1,2,3,1,2, 2,1,3,1,1,1,3)
NRELP(J) J = 1,14	The initial number of releases for node J.	(0,1,1,2,3,1,2, 2,1,3,1,1,1,3)
RESA(I) I = 1,4	The number of units of resource I available during the simulation.	(4,3,5,4)
RESAP(I) I = 1,4	The initial number of units of resource I available.	(4,3,5,4)
RESUT(I) I = 1,4	The utilization of resource I for the simulation.	0
REUSE I = 1,4	The time-integrated utilization of resource I.	0

```
      OUTPT: PROCEDURE;
      DCL ATRIB(*) BIN FLOAT CTL EXT;
      DCL (NA(16),NATRIB) FIXED BIN(31,0) EXT;
      DCL PRINTQ ENTRY(FIXED BIN(31,0));
      DCL (I,IBEG,IEND,K,NADJST) FIXED BIN(31,0) STATIC;
/*                                                                    */
/****PROCEDURE OUTPT IS A SUBPROGRAM WHICH IS CALLED BY THE          ****/
/****GASPL/I CONTROLLING ROUTINE.  IN THIS EXAMPLE, IT HAS BEEN      ****/
/****WRITTEN TO GENERATE REPORTS FOR EACH RESOURCE, LISTING          ****/
/****NODE ACTIVITIES RANKED BY RESOURCE REQUIREMENT.                 ****/
/*                                                                    */
 F1: FORMAT(PAGE,SKIP(2),X(10),A,SKIP(2),X(7),A);
      NA(14),NA(15),NA(16)=1;
      NATRIB=7;
      PUT EDIT('**  SYSTEMS ANALYST     **',
               'RESOURCE REQUIREMENTS BY NODE')(R(F1));
      NADJST=1;
      IBEG=2;
      IEND=15;
      CALL PRINTIT;
      PUT EDIT('** MARKETING PERSONNEL **',
               'RESOURCE REQUIREMENTS BY NODE')(R(F1));
      NADJST=15;
      IBEG=16;
      IEND=29;
      CALL PRINTIT;
      PUT EDIT('** MAINTENANCE PERSONNEL **',
               'RESOURCE REQUIREMENTS BY NODE')(R(F1));
      NADJST=29;
      IBEG=30;
      IEND=43;
      CALL PRINTIT;
      PUT EDIT('** ENGINEERING PERSONNEL **',
               'RESOURCE REQUIREMENTS BY NODE')(R(F1));
      NADJST=43;
      IBEG=44;
      IEND=57;
      CALL PRINTIT;
      RETURN;
/*                                                                    */
/****PROCEDURE PRINTIT IS AN INTERNAL PROCEDURE CALLED TO            ****/
/****DO THE ACTUAL PRINTING ONCE THE VARIOUS PARAMETERS              ****/
/****HAVE BEEN SET.                                                  ****/
/*                                                                    */
      PRINTIT: PROCEDURE;
      DO I=IBEG TO IEND;
       K=I-NADJST;
       PUT EDIT('NODE',K,'START','END','ACTIVITY','SYSTEMS','MARKETING',
               'MAINTENANCE','ENGINEERING',(2) 'NODE','NUMBER',
               'ANALYST',(3) 'PERSONNEL')
               (SKIP(3),X(20),A,F(3),A,SKIP(2),X(3),A,X(8),A,X(10),A,
               X(5),A,X(6),A,X(4),A,X(2),A,SKIP,X(3),A,X(9),A,X(9),
               A,X(7),A,X(6),3 (A,X(4)));
       CALL PRNTQ(I);
      END;
      RETURN;
      END PRINTIT;
      END OUTPT;
```

Figure 4

PL/I Listing of Procedure OUTPT

References

1. Pritsker, A. Alan B. and Philip J. Kiviat, Simulation with GASP II, Prentice-Hall, Inc., 1969.

2. Young, Robert E., "GASPL/I--The Conversion and Extension of GASP II from FORTRAN to PL/I", Unpublished Masters Thesis, Purdue University, December, 1972.

3. Purdue University Computing Center, "GASPU (Improved Package of the GASP II Programs--Purdue University Version)," Document H 4--GASPU, September 1971.

4. Pritsker, A. Alan B. and Nicholas R. Hurst, "GASP IV: A Combined Continuous-Discrete FORTRAN Based Simulation Language", SIMULATION, September 1973, pp 71-75.

5. Pritsker, A. Alan B., The GASP IV Simulation Language, To be published by John Wiley & Sons, Inc., July 1974.