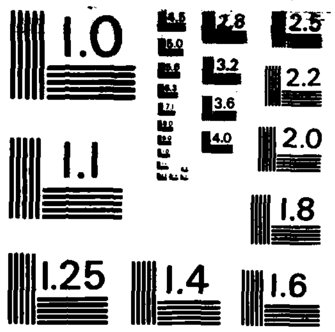
AD-A141 127 AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER 1/2
AIDED DESIGN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON
AFB OH SCHOOL OF ENGI.. G M ANDERSON ET AL. DEC 83
UNCLASSIFIED AFIT/GOR/OS/83D-2 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

1

AFIT/GOR/OS/83 D-2

AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER AIDED DESIGN

THESIS

Gary M. Anderson
Captain, USAF

Chris R. Commeford
Captain, USAF

AFIT/GOR/OS/83D-2

unclassified
SECURITY CLASSIFICATION OF THIS PAGE

A148 127

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS none
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GOR/OS/83D-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering Air Force Inst. of Tech. — 6b. OFFICE SYMBOL *(If applicable)* AFIT/EN	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State and ZIP Code)* Wright-Patterson AFB, OH 45433	7b. ADDRESS *(City, State and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State and ZIP Code)*	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE *(Include Security Classification)* See Box 19	

12. PERSONAL AUTHOR(S)
Gary M. Anderson Capt USAF Chris R. Commeford Capt USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
MS Thesis	FROM ____ TO ____	1983/Dec/1	147

16. SUPPLEMENTARY NOTATION

Approved for public release: IAW AFR 190-17.
[illegible] 7 May 84
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
12	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Automated Programming, Computer Aided Design, Computer Graphics, Minicomputers, Simulation Languages

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER AIDED DESIGN (unclassified)

Advisor: Lt Col Peter Bobko

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED [✓ ☒] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Peter Bobko Lt Col USAF	22b. TELEPHONE NUMBER *(Include Area Code)* (513) 255-3362 — 22c. OFFICE SYMBOL AFIT/EN

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

unclassified
SECURITY CLASSIFICATION OF THIS PAGE

Abstract

The Pascal computer program developed within, uses an Apple microcomputer and Graphics Tablet to allow an analyst to create a Q-GERT simulation network using computer aided design techniques. The analyst need only select commands from the Graphics Tablet's programmed menu and answer questions regarding the symbols to be drawn. The program will display the network on the high resolution graphics screen and generate a text file containing the computer source code for input to the Q-GERT Analysis program residing on a mainframe computer. The basic concepts of Q-GERT are incorporated into the program. This includes source, regular, queue, statistic, and sink nodes along with activities and parameters. The program was written to be expandable for further development to include more complex concepts of Q-GERT.

AFIT/GOR/OS/83D-2

AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER AIDED DESIGN

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for a

Masters Degree

Gary M. Anderson
Captain, USAF

Chris R. Commeford
Captain, USAF

December 1983

Preface

Accessibility of microcomputers has brought increased computing power to an analyst's fingertips. The purpose of this research is not to teach Q-Gert, but, to give an analyst familiar with Q-Gert a useful tool making simulation convenient and error free. For a complete description of Q-Gert we refer you to Modeling and Analysis Using Q-Gert Networks by A. Alan B. Pritsker. If an analyst has available to him an Apple computer and graphics tablet, using our program, he can create basic Q-Gert simulation models. The network diagram is displayed on a high resolution graphics screen and the program will generate a text file containing the computer source code that can be sent via telephone to a mainframe computer to run the Q-Gert Analysis program.

> [illegible] vii

A major side effect of any thesis is the learning process of conducting research. We learned a great deal about the methodology of research and, equally as important, how to work as a team. With the guidance of our thesis committee, LtCol. Bobko and Professor Richard, we learned more about the Apple computer and Pascal then either of use dreamed possible. Having no previous Pascal programming experience, we spent many long hours agonizing over seemingly trivial problems. LtCol. Bobko was our guiding light helping to solve many undocumented peculiarities of Apple Pascal; while, Professor Richard gave us valuable insight to the available data structures we needed to store all the information to create the source code. Many thanks

to the both of them.

We must also thank our families; Pam and Tricia Anderson, and Susan and Donald Commeford for their patience and endurance when we were immersed in our studies.

Gary M. Anderson
Chris R. Commeford

Table of Contents

List of Figures

from pg: ii

Abstract

The Pascal computer program developed within, uses an Apple microcomputer and Graphics Tablet to allow an analyst to create a Q-Gert simulation network using computer aided design techniques. The analyst need only select commands from the Graphics Tablet's programmed menu and answer questions regarding the symbols to be drawn. The program will display the network on the high resolution graphics screen and generate a text file containing the computer source code for input to the Q-Gert Analysis program residing on a mainframe computer. The basic concepts of Q-Gert are incorporated into the program. This includes source, regular, queue, statistic, and sink nodes along with activities and parameters. The program was written to be expandable for further development to include more complex concepts of Q-Gert.

AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER AIDED DESIGN

I. BACKGROUND

Simulation has been used for many years to understand the behavior of systems. It has been applied to a variety of manufacturing, service, and defense industries to evaluate various strategies for achieving goals or solving problems. Many times direct experimentation is inappropriate when trying to solve problems. It may be disruptive to the organization, costly, time consuming, or impossible to explore an alternative in real life experimentation. Therefore, simulation may be appropriate if no complete mathematical formulation of the problem exists, or the analytica[illegible] method needed is so complex that simulation is a simplier solution method. Simulation models also allow the experimenter to develop an intuitive understanding of the system and 'feel' for the problem. McKenney [12] had the following to say about using a simulation model to understand the real system:

> When the manager had achieved a viable understanding and began to manipulate the model, he cor[illegible]inuously gained new insights in[illegible] is operatic[illegible] He desired the model to tes[illegible] a[illegible]ariety of al[illegible]atives so he could evaluate [illegible]e new insi[illegible]. In essence, he was using the model to am[illegible]y his manipulative skill by explicitly id[illegible]fying all important ramifications of a given change. Because of the complexity of the system it may have been possible for him to do this on the real system, but very tedious, and he probably

would have made errors. Thus, he turned to the model as an evaluator of his new insights. It is conjectured the model design will never be stabilized, but continue to develop in response to the manager's new understanding.

Schoderbek,Schoderbek,and Kefalas [16] posed a systems science paradigm which can be used for problem solving with simulation. First, conceptualization, where the system being studied is defined, boundaries set, restrictions applied, and measures of effectiveness identified. The system is analyzed as to its place and interaction within it's environment. An output of the conceptualization phase of simulation is a flow diagram linking all the major pieces into an accurate picture of the working system. Next, the analysis and measurement phase wherein the system is expressed in quantifiable terms enabling measurement of the changes that will occur when the system is manipulated. This phase also entails structuring the problem solving technique to measure and analyze the changes under different conditions imposed on the system. Finally, computerization involves describing the system using a simulation language, the source code, and running this code on a compatible computer to generate results. These three steps, are iteratively refined as the simulation model is developed. For instance, while translating the model into a computer source code, a problem may arise in the logic flow of the original model. The analyst must go back to the conceptual-ization phase and attempt to redesign or reformulate the model and go through the 3-phased process again [17].

A popular simulation language is Q-Gert, developed by Dr. Alan B. Pritsker as a "network modeling vehicle and computer analyst tool [14]." The language is a generalization of PERT and an extension of GERT, two specialized simulation languages, enabling queuing and decision capabilities to be analyzed by computer simulation. Q-Gert has been successfully used in studying manufacturing, service and defense systems. "It provides engineers, business analysts and operation researchers with a graphical vehicle for modeling, analysis, and communication [14]." Q-Gert networks are models of systems consisting of activities, services, and queues. Through computer simulation using Q-Gert, problem solving and risk analysis can greatly enchance the accuracy and reliability of information available to the decision maker.

Q-Gert is used in all three phases of the system science paradigm. First, in the conceptualization phase, the logic flow layout of the system is represented using Q-Gert symbols which specifies relationships between parts of the systems and the overall flow pattern through the system. The logic flow diagram is in a form enabling the analyst to communicate to others his concept of the system's structure. Feedback from the decision maker and acquired knowledge about the system, may necessitate a change to the model. This change may entail redefining relationships, thus, changing the logic flow diagram to reflect the modified version of the system. In the analysis and measurement phase, the Q-Gert language allows specification of distributions defining the transaction flow patterns through the system, ie. rates of flows between services, combinations of servers at activities, or queue space available prior to an activity. Finally, the graphical symbology of the network is manually translated into Q-Gert computer source code to be run on a computer system able to handle the entire Q-Gert Analysis program.

From the authors' personal experiences, the translating of the symbology to computer code is tedious, time consuming and often riddled with input errors. Through the refinement process, even if the computer source code is accurate for the original model, incorporating any changes will again be susceptible to input errors and the resultant time spent debugging the computer source code.

Graphics, Simulation and the Microcomputer

Computer aided design (CAD) has been used by engineering disciplines since the middle 1960's. Architectural engineers use CAD to design building layouts incorporating other computer analysis programs to evaluate layouts against existing regulations, and building codes [21]. With the ability to change layouts, testing different room sizes, window and door placements and even construction materials, the designers can 'optimize' the layouts. Using CAD in conjunction with analysis programs that compute material and labor costs enables the architectural engineers to accurately estimate the total cost of a design. Utilizing an editing capability, changes can be made to a design and the analysis accomplished in minimal time [4]. Circuit designing also prospered through the use of CAD. Electrical engineers conceptualize circuits then simulate their potential capabilities with additional analysis programs [8]. Time is a vital commodity saved by using CAD. Another advantage that emerged by combining CAD and analysis programs is the ability to construct and test complex structural designs using only computers without having to actually build a scale model or full scale prototype for testing, again saving time and money [21]. The use of CAD in numerous engineering disciplines is expanding rapidly; however, CAD's use by operation researchers is virtually nonexistent.

Use of computer graphics by operation analysts is predominantly the depiction of results of the analysis consisting of graphs, charts and tables [22]. Computer graphics has not been used to develop the computer models in the conceptualization phase. The capability to instantly express visually the analyst's ideas would save both his and the client's time. Enhanced communications through quick pictorial representation of the model being analyzed also helps eliminate discrepancies between the client's and analyst's concept of the system [14]. This interaction enables a better model to be developed and helps the client see how his system 'really' functions.

The front end use of computer graphics in operations research techniques is limited although its potential use in simulation is most pronounced. This research will develop a useful tool for modeling systems using the symbolic Q-Gert simulation language. Though use of microcomputers, we will make this tool accessible to the analyst and easy to use.

Why Microcomputers

Today's operations researchers must have access to a computer to effectively analyze complex problems. Often the ability to utilize mainframe computers is cumbersome, due partially to limited terminal access, making modeling and analyzing a slow process [7]. With the advent of the micro computer, the gap between simple calculators and mainframe computers has been bridged [18]. The micro computer's flexibility allows the computer to be used as a stand-alone system for solving small problems or as an intermediary to a mainframe computer acting as a terminal linked to the larger system. Used as a remote terminal, ready to run 'source' programs developed on the microcomputer are sent to the mainframe computer for execution. The main frame computer with its large memory capacity can run the program and send the results back to the microcomputer or print the output at a central site. The linkage between the microcomputer can be accomplished through telephone lines or a direct connect system. With a microcomputer at his disposal, the analysts' desk becomes a complete work station with the accessibility of a calculator and the power of a mainframe computer at his disposal [1].

Our purpose is to use the microcomputer as an intermediary between the analyst and the mainframe computer in simulation applications. Advantages of using a microcomputer includes minimizing mainframe computer usage in the designing, editing, and debugging stages of programming [18]. A mainframe computer is not needed to process the graphics and generate source code, a microcomputer can handle this task. The computer source code can then be sent for execution to a mainframe computer when complete or at a later time.

Simulation and Miccocomputers

"Simulation is the development and use of models to aid in the evaluation of ideas and the study of dynamic systems and situations [13]." Presently the use of microcomputers in simulation is limited to small repetitive analytical problems. Each model developed is for a specific analysis and is composed of mathematical expressions and manipulation [15]. There is not a powerful simulation analysis program such as Q-Gert's available to run on a microcomputer because of the lack of memory capability that large simulation programs demand.

Graphics and Simulation

"A picture is worth a thousand words" has ample applicability to engineering work. A schematic is certainly more meaningful for use in communicating ideas between engineers and managers than a complicated mathematical formula explaining the same ideas. The schematic can focus attention on relationships and interdependencies more quickly. A manager not familiar with the intricacies of model building can visually comprehend the ideas being explained.

In simulation, computer generated graphics is used only in the presentation of output [9]. Histograms, graphs and bar charts are used to explain the results of the simulation. Understanding the output of the model is paramount to a decision maker, but a graphic representation of the system being modeled is equally important to his understanding of why the particular results were obtained. Computer graphics incorporated as a means of input to the simulation program will give the decision maker a schematic of the system as well. Computer graphics can be used effectively for conceptualizing system designs, communicating with clients, as well as, input to a computer analysis program.

II. Statement of the Problem

The foundation of any Q-Gert simulation is the network depiction of the flow of transactions through the system. As changes or alternatives are considered, the analyst should modify the graphical representation of the network then change the computer source code. The manual translation of Q-Gert symbology into the source code frequently results in errors due to the rigorous input requirements of the Q-Gert language. The ability to develop and then change a network's graphic display with automatic generation of the corresponding computer source code necessary to run analysis programs would create substantial time savings for an analyst.

Research Questions

This research effort will attempt to eliminate the basic problems discussed above answering the following:

1. Using computer aided design techniques, is there a way of placing Q-Gert symbols on a microcomputer screen using a graphics tablet?
2. Using an appropriate programming language, is it possible to automatically generate Q-Gert computer source code from the graphical representation created in question 1?
3. Is it possible to edit a previously drawn network and incorporate the same changes to the computer source code?

Research Objectives

The overall objective is to successfully execute a Q-Gert model whose computer source code was automatically generated using computer aided graphics. The source code generated will be sent to AFIT's CYBER computer system to run the Q-Gert Analysis program.

The research sub-objectives are:

1. Using a graphics tablet with an Apple II+ microcomputer, represent a Q-Gert network on the computer screen.

2. Using Pascal, convert the graphical representation on the screen into a text file containing the complete and accurate Q-Gert source code.

3. Edit a previously drawn Q-Gert network and update the appropriate text file containing the source code automatically. This will necessitate a save and reload capability.

4. Send a text file to the CYBER and successfully execute the simulation model.

It would be desirable to include the full range of Q-Gert symbols in this effort. However, due to the complexity of some symbols and the limitation of time, not all of the symbols may be included. It is the intent of this effort to achieve all of the afore mentioned objectives on at least a subset of Q-Gert symbols.

Materials

This research effort will use computer aided design to draw a Q-Gert network on the high resolution screen of an Apple II+ microcomputer. At the same time, Pascal is used to develop a data base which will be partially built by the user answering questions presented on another screen. Finally, the data base will be manipulated to produce the finalized source code which can be sent via telephone lines to the CYBER mainframe computer which will run the Q-Gert Analysis program.

A listing of the equipment needed to run this program is as follows:

1. Apple II+,64K RAM, two disk drives
2. M & R Enterprises Sup'r'terminal board in slot #3
3. Apple Graphics Tablet in slot #5
4. Apple Pascal operating system
5. Two monitors
6. Modem to communicate with a mainframe computer

Graphics Tablet.

The Apple Graphics Tablet is a device that converts the position and movement of a special pen into numbers which the computer can understand. The software associated with the tablet tells the Apple computer how to draw shapes, lines, and letters on the high resolution graphics screen using the information input from the tablet. These programs turn the combined computer and tablet system into a sketchpad or engineer's drawing board. This particular graphics tablet is primarily designed to work in a BASIC language environment utilizing Applesoft programs for displaying the drawings on the high resolution graphics screen [10]. In this capacity, there can be no mixing of graphics and text needed to completely illustrate a Q-Gert network without additional software. Apple Pascal, however, has this capability with Turtlegraphics.

Pascal.

In addition to the drawing functions inherent in Apple Pascal, a sophisticated computer language is needed capable of handling the task at hand. Due to the anticipated size of the program and type of data structure needed, Apple Pascal was considered the appropriate computer language to use. Apple Pascal has two memory conserving capabilities that are essential to execute large programs on microcomputers with a limited amount of memory available.

After the Apple Pascal system is loaded into the computer, the memory available to execute a Pascal program on the Apple II is 20k of random access memory (RAM). Apple Pascal has structures available to the user called units that need not reside in memory unless they are being used [2]. Each unit is a collection of procedures and functions compiled separately from the main program. Unlike standard Pascal procedures or functions, a unit can exist separately from the body of the main program text and still be linked to a Pascal program's object code at run time. The power of a unit lies in its ability to house multiple procedures or functions, built in Pascal or assembly language, under one roof. All of these procedures and functions are available from within a Pascal host program. Additionally, units may be nested within each other. By developing the program in parts, each part can be made a unit; and, when a procedure or function within a unit is needed by the main program, only then will the unit come into the computer's memory thus minimizing the memory required to run the program. When the unit is no longer needed, it can be removed from memory until it is needed again.

The other major reason for selecting Pascal is the use of the dynamic data structure called linked lists used in conjunction with record data types [6]. A record is similar to an array in that they both can represent a group of elements with a common name. However, while the elements of an array must all be of the same data type; integer, real, string, etc., the elements of a record may be a mixture of data types. Within the same record, one field may be designated real while another may be designated as a string or even another record. Individual fields of a record can be accessed and similar records can be linked together for sequential access through a pointer system.

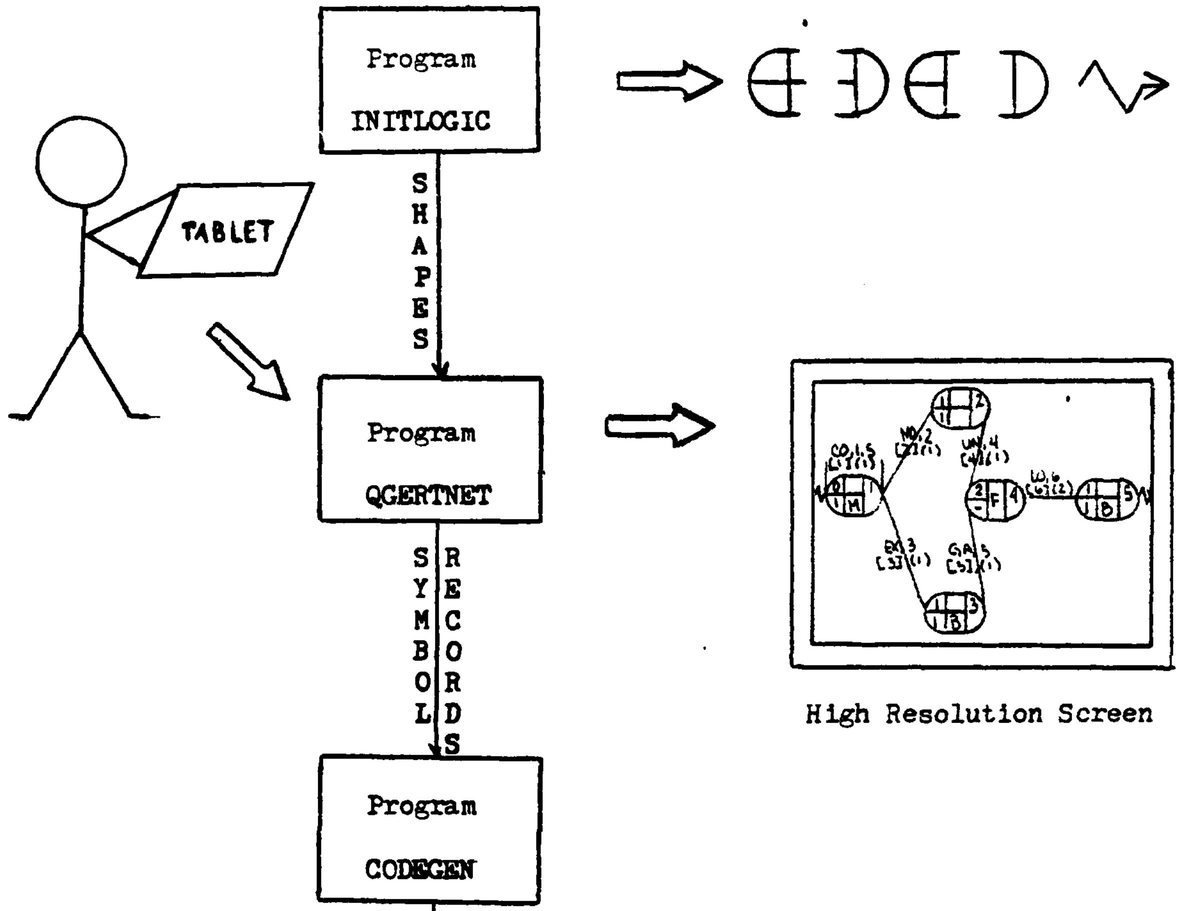
The use of arrays, that are fixed in dimension, sets a limit on the size of the network. Alleviating this limitation, the linked list data structure is expandable as needed. For instance, if a network initially had 5 regular nodes the number of components in the linked list would only be 5. If the network was expanded, additional components would be added to the linked list only as required. If arrays were used, memory would be allocated at the outset of program execution. If the dimension of an array were set at 50 nodes but only 5 were used, the memory reserved would still accommodate 50 nodes that the array specifies. This may necessitate limiting the size of the program or units called by the program during execution. The linked list data structure is available in Apple Pascal and saves memory over the conventional array structure.

III. PROGRAM DEVELOPMENT

The concept of jointly using a graphics tablet and a microcomputer is not new. Dan Sokol's article [19] outlines his use of interactive graphics for developing electrical circuit schematics. Our program goes beyond just schematic displays, it will create a Q-Gert network and generate Q-Gert source code from the user's graphical selections and solicited inputs. The first step involved interfacing the graphics tablet and the Pascal operating system enabling information to be passed between the two. Once accomplished, the programming to accomplish the stated objectives was started.

The system was developed in three parts. First, the Q-Gert symbols to be displayed were developed in program INITLOGIC. Then, program QGERTNET was developed to draw the network; and, create and edit the data structure. Finally, program CODEGEN transforms the information contained in the data structure to a text file consisting of the desired computer source code. Figure 1 depicts the macro view of the source code generation system. INITLOGIC creates a file of shapes which QGERTNET accesses to draw the network on the high resolution screen. Then, QGERTNET creates files of symbols' records which CODEGEN accesses to generate the Q-Gert source code.

Two programming concepts built into the system are expandibility and user friendliness. The primary requirement to permit future expansion is the conservation of computer memory. As the program grows, available computer memory



```
GEN,USER,NET,1,1,1984,1,1,10,50,5,E*
SOU,1,0,1,D,M,L*                         SOURCE NODE
REG,2,1,1,D,,L*                          REGULAR NODE
STA,3,1,1,D,B,N,N,L*                     STATISTIC NODE
QUE,4,2,,D,F,N,N*                        QUEUE NODE
ACT,1,1,CO,1.5,1,1*                      ACTIVITY #1
ACT,1,2,NO,2,2,1*                        ACTIVITY #2
ACT,1,3,EX,3,3,1*                        ACTIVITY #3
ACT,2,4,UN,4,4,1*                        ACTIVITY #4
ACT,3,4,GA,5,5,1*                        ACTIVITY #5
ACT,4,5,LO,6,6,2*                        ACTIVITY #6
PAR,2,3.0,0.0,,0.8*                      NORMAL DIST
PAR,3,2.5*                               EXPONENTIAL
PAR,4,,1.0,3.0*                          UNIFORM DIST
PAR,5,3.0,0.0,10.0,0.5*                  GAMMA DIST
PAR,6,4.0,2.0,6.0,0.2*                   LOGNORMAL
SIN,5,1,1,D,I,N,N,L*                     SINK NODE
FINISH*
```

Source Code

Figure 1. Macro View of Source Code Generation System.

decreases unless steps are taken to minimize the required memory to execute the program. To conserve memory, most procedures are contained in units which remain in computer memory only when a procedure within the unit is being executed. Also, the program is user friendly. Through timely and informative prompts, the user can quickly develop a network and generate the source code. For convenience, an editing capability was written into program QGERTNET. This facilitates changing the network design, as well as, correcting accidental input errors. When the user is satisfied with the network developed, the data structure can be saved on diskette to redraw the network or generate the source code at a later time.

The following discussion will briefly explain the data structure, layout and setup of the Apple Graphics Tablet followed by the the logic that went into developing each of the three programs.

Data Structure

Pascal has a mechanism for creating dynamic variables which are defined at compile time but only created during execution of the program. Program QGERTNET declares records as dynamic variables. Different types of records are declared corresponding to the different Q-Gert symbols. Source and regular nodes require the same information to describe their function and parameters to the Q-Gert Analysis program; therefore, these symbols' records are of the same type. Statistic and sink nodes are also similar to each other while queue nodes, activities, and parameters require separate record types to store their information. Each record also contains information to specify the symbol and its location on the high resolution graphics screen. Figure 2 depicts the fields within each type of record.

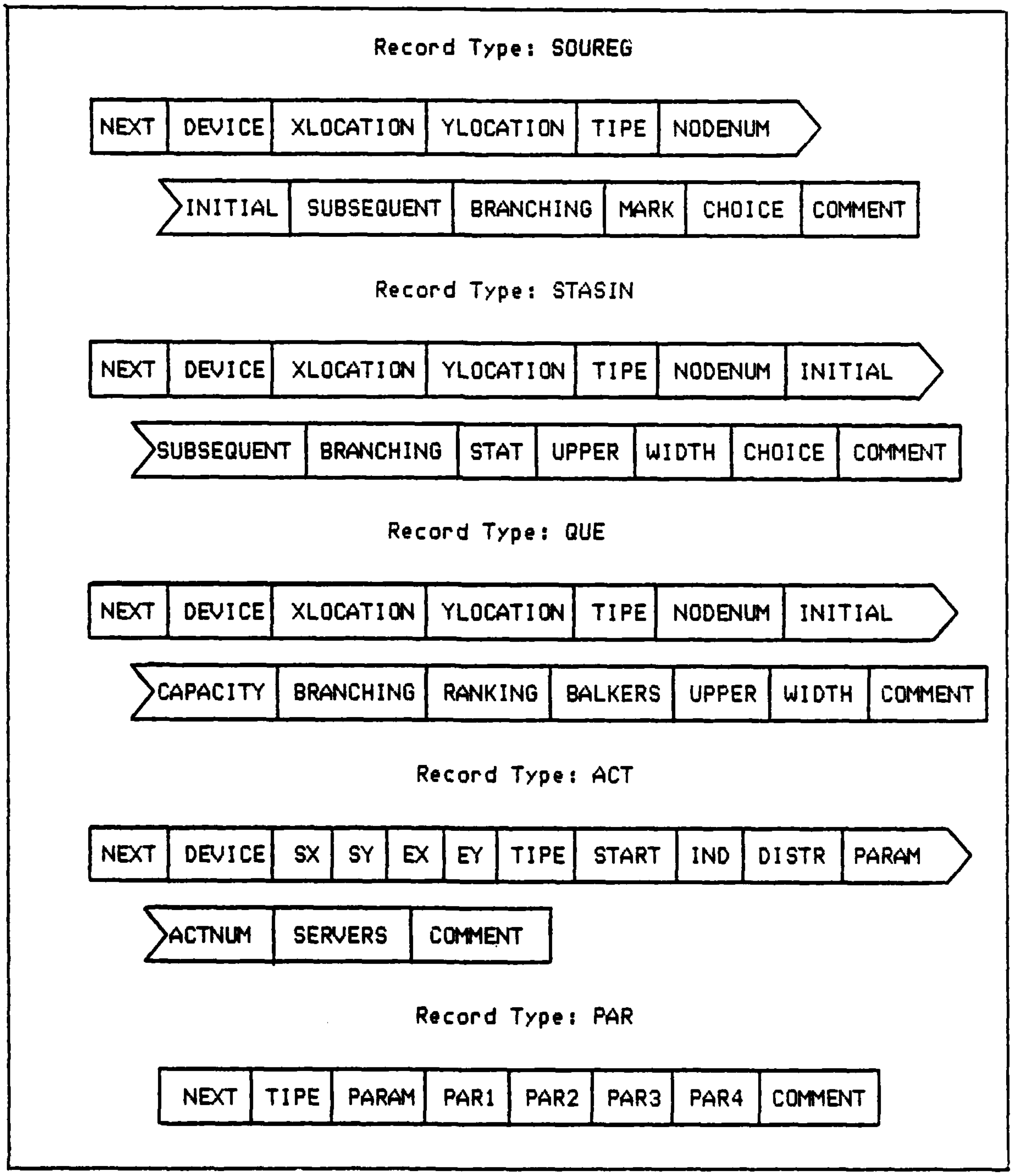

Record Type: SOUREG
NEXT | DEVICE | XLOCATION | YLOCATION | TIPE | NODENUM
INITIAL | SUBSEQUENT | BRANCHING | MARK | CHOICE | COMMENT
Record Type: STASIN
NEXT | DEVICE | XLOCATION | YLOCATION | TIPE | NODENUM | INITIAL
SUBSEQUENT | BRANCHING | STAT | UPPER | WIDTH | CHOICE | COMMENT
Record Type: QUE
NEXT | DEVICE | XLOCATION | YLOCATION | TIPE | NODENUM | INITIAL
CAPACITY | BRANCHING | RANKING | BALKERS | UPPER | WIDTH | COMMENT
Record Type: ACT
NEXT | DEVICE | SX | SY | EX | EY | TIPE | START | IND | DISTR | PARAM
ACTNUM | SERVERS | COMMENT
Record Type: PAR
NEXT | TIPE | PARAM | PAR1 | PAR2 | PAR3 | PAR4 | COMMENT

Figure 2. Record Fields

Associated with dynamic variables are pointers. These variables contain the address in memory of the dynamic variable it references. Records are accumulated into lists by having each record contain a link or pointer to the next record in the list. Using these dynamic variables, a data structure called a linked list is built that can expand or contract as the program executes. The programer does not have to know in advance how long the list will be. The only size limitation is the amount of available computer memory. During program execution using this data structure, the user can create new records when building a network, change record fields when editing, or destroy records when modifying the network.

Each type of record is kept in a separate linked list. Reference to a record within a linked list is not by name but by a pointer. To keep track of the records, unique pointers, BASE(i) and NEXT(i), are declared for each list. NEXT(i)^ is the record within the particular linked list referenced by pointer NEXT(i). BASE(i) is always reset to point to the first record in the list or has the value of NIL, if the list is empty. NEXT(i) is used to traverse the list to locate a particular record. Additionally, when the program creates a new record, it is added to the appropriate list and linked to the other records by a pointer field (NEXT(i)^.NEXT) inherent in each record. Figure 3 illustrates the addition of a new records to a list.

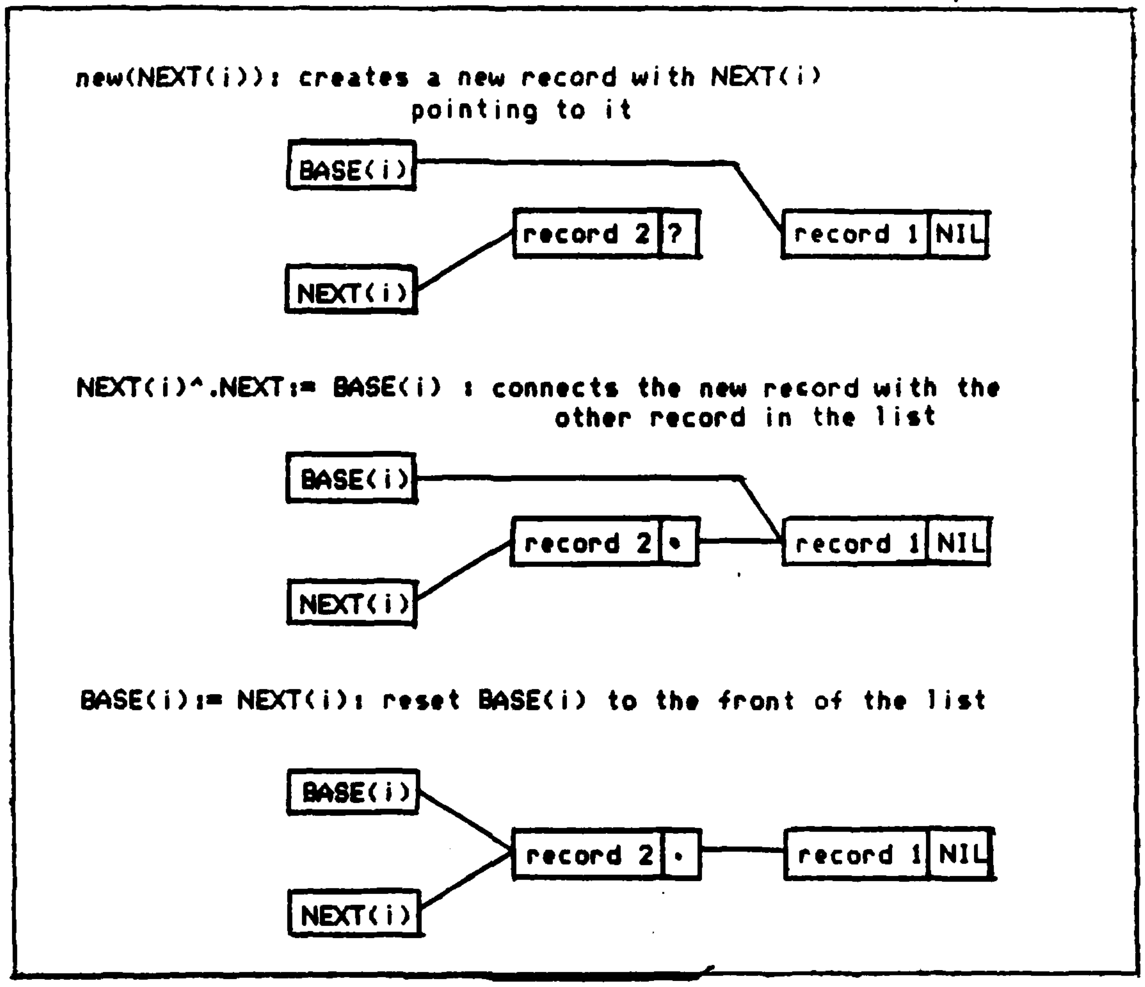

new(NEXT(i)): creates a new record with NEXT(i) pointing to it
BASE(i)
record 2
?
record 1
NIL
NEXT(i)
NEXT(i)^.NEXT:= BASE(i) : connects the new record with the other record in the list
BASE(i)
record 2
record 1
NIL
NEXT(i)
BASE(i):= NEXT(i): reset BASE(i) to the front of the list
BASE(i)
record 2
record 1
NIL
NEXT(i)

Figure 3. Adding a Record to a Linked List.

To find a particular record within a linked list, the user must specify the field and the field's value to identify the correct record. NEXT(i) initially points to the first record in the list. If the specified field's value of this record is not equal to the desired record's then NEXT(i) is reset to point to the next record in the list. To ensure that the entire list is interrogated, after searching the first record, the second record is searched prior to advancing the pointer. The search continues through the list until the desired record is found or the interrogated record's pointer field is NIL. Figure 4 illustrates the search routine to find record 2 whose field called 'here' is equal to element.

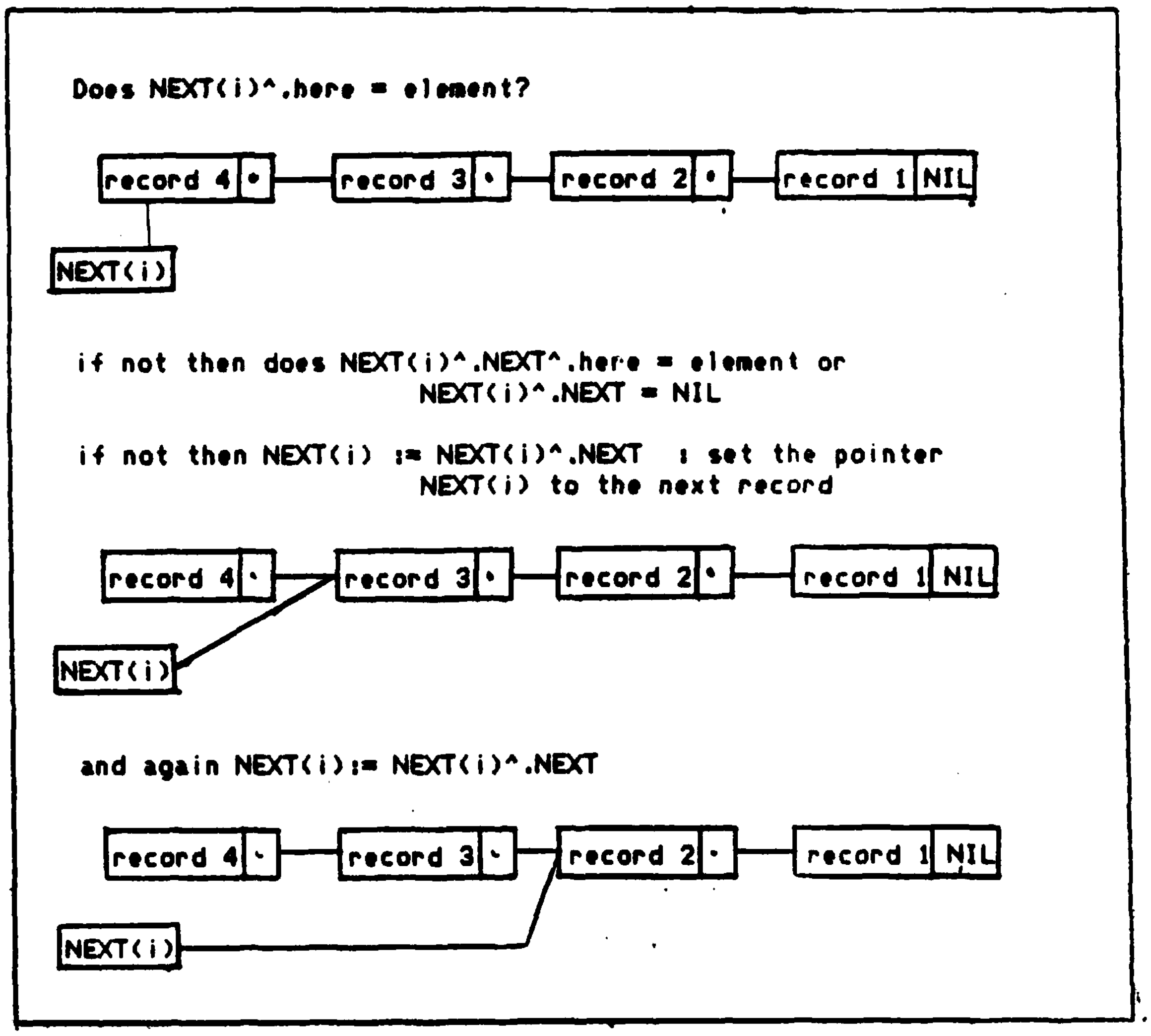

Does NEXT(i)^.here = element?
record 4
record 3
record 2
record 1
NIL
NEXT(i)
if not then does NEXT(i)^.NEXT^.here = element or NEXT(i)^.NEXT = NIL
if not then NEXT(i) := NEXT(i)^.NEXT ; set the pointer NEXT(i) to the next record
and again NEXT(i):= NEXT(i)^.NEXT

Figure 4. Traversing a Linked List.

Pointers can also facilitate deleting records within the linked list. To locate the record to be deleted, the same routine just described is used except the pointer NEXT(i) is set to the previous record in the list. The pointer field of this record is reset to point to the record following the record to be deleted (NEXT(i)^.NEXT^.NEXT). When the linked lists are saved to files, only the records that are linked in succession are retained, circumventing the record to be deleted. Figure 5 illustrates deleting the second record in a linked list.

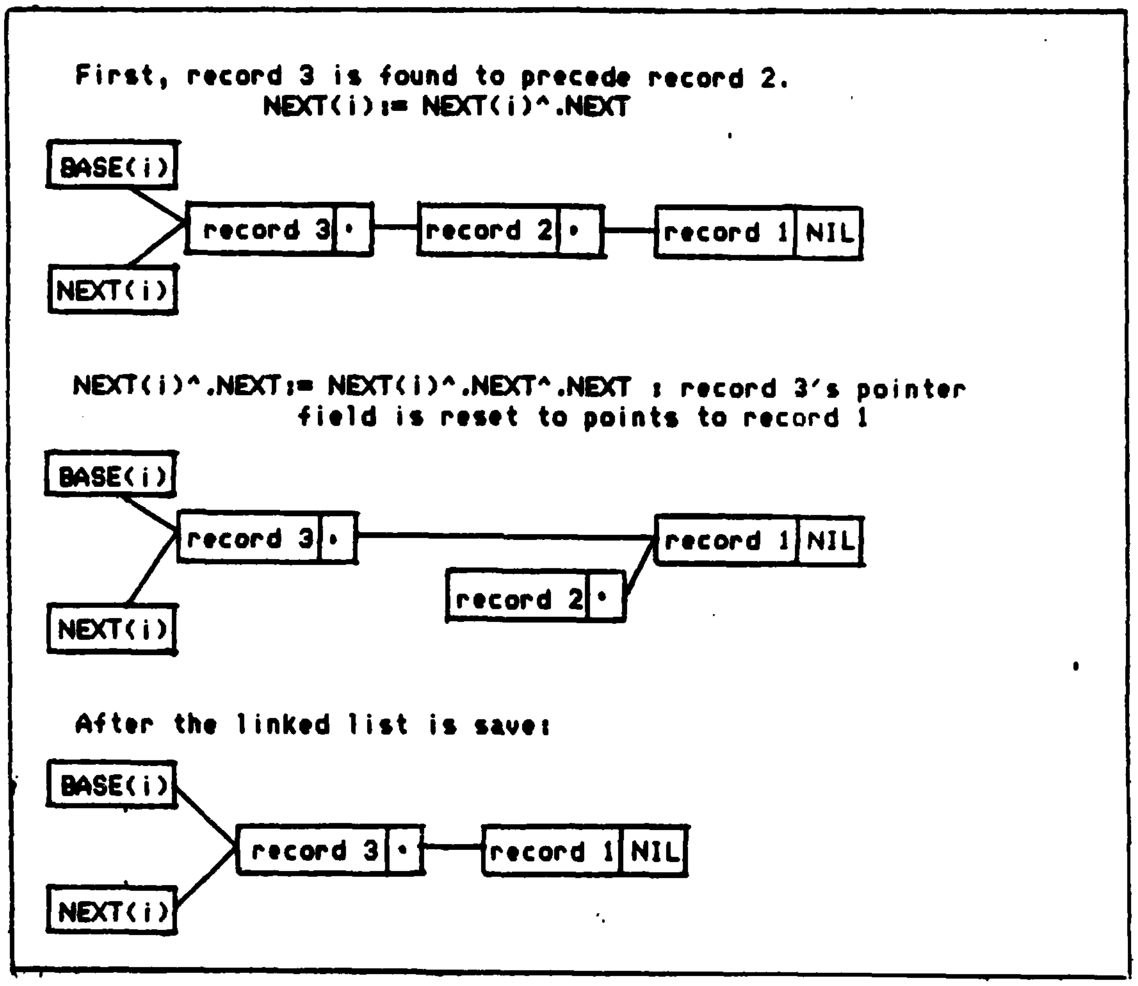

First, record 3 is found to precede record 2.
NEXT(i):= NEXT(i)^.NEXT
BASE(i)
record 3
record 2
record 1
NIL
NEXT(i)
NEXT(i)^.NEXT:= NEXT(i)^.NEXT^.NEXT ; record 3's pointer field is reset to points to record 1
BASE(i)
record 3
record 1
NIL
record 2
NEXT(i)
After the linked list is save:
BASE(i)
record 3
record 1
NIL
NEXT(i)

Figure 5. Deleting a Record from a Linked List.

The the linked list data structure was selected over an array structure because it takes up only as much memory as needed to store the Q-Gert network information. The programmer does not have to specify a limit on the size of the network, as in arrays. Also, this structure is convenient for accessing particular records and their associated fields when editing and generating computer source code.

Graphics Tablet Setup

To use the Apple Graphics Tablet, it must be able to communicate with the Pascal operating system. Sokol included in his article [19] the assembly language program that made the two compatible. The reason that the tablet would not interface to Pascal was the Pascal BIOS (basic input/output subsystem) did not recognize the Graphics Tablet's existence. The program PAD.ASSY.TEXT (Appendix A) is the assembly language linkage routine which reads data from the pad and transfers it back to Pascal. The program contains two procedures: SETUPAD which sets the tablet's default parameters; and, READPAD which reads the pad, flashes the cursor, and scales the results. Additionally, SETUPAD initializes the interface between the two screens. The symbols are dislayed on one screen using a 40 column display capability while all text is displayed on the other screen utilizing the sup'r'terminal's 80 column capability. Internal to the procedure READPAD is a requirement to store x and y coordinates read from the graphics pad in decimal locations 645 thru 648. Pascal recovers this data using PEEK and POKE commands added to Pascal by a unit (Appendix B) also written by Sokol [20]. Once data could be read from the graphics tablet, the main programs could be written.

Graphics Tablet Layout

The Apple Graphics Tablet has a mylar overlay that is placed on the surface of the tablet. The overlay divides the surface of the Tablet into different areas, each area having a different meaning. The overlay is aligned using the Graphics Tablet software's MENU ALIGNMENT program which is written in BASIC [10]. Changing to Pascal necessitates redefining the functional areas. This is done in SETUPAD. The center of the overlay (Figure 6), the active area, represents the high resolution graphics screen. The bottom three rows of the Tablet are programmed as a menu for selecting symbols to draw or commands to execute.

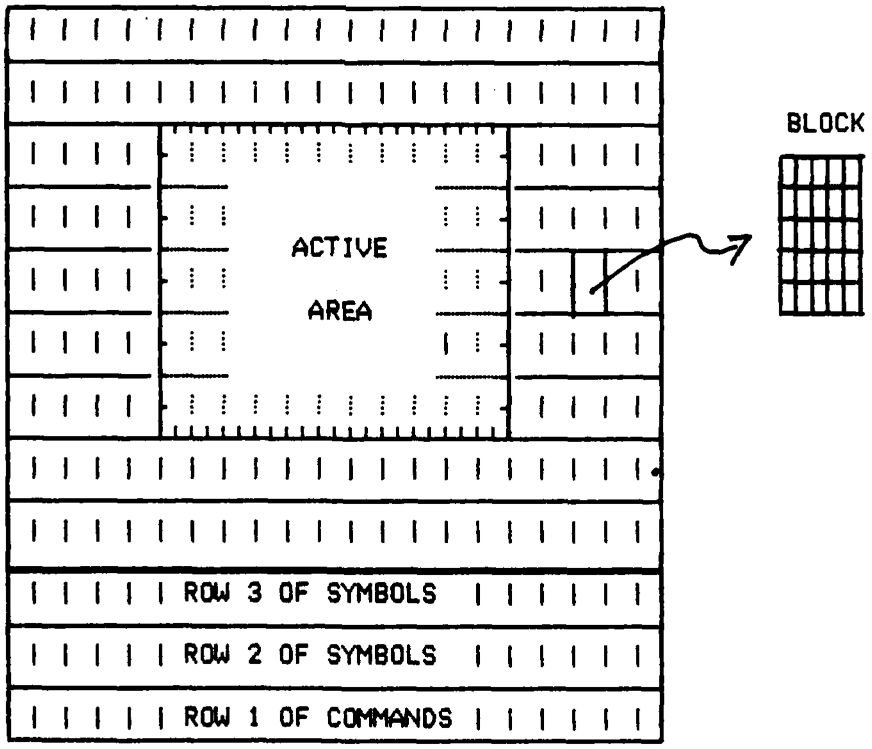


Figure 6. Graphics Tablet's Functional Areas.

Program INITLOGIC

The program INITLOGIC creates the shapes used by program QGERTNET to draw the Q-Gert symbols. INITLOGIC converts groups of strings to boolean arrays and saves the arrays on disk in a file called LOGIC.CHARSET. Each named shape is a square array, 21 elements on a side. The symbols must be large so text can be printed inside their boundaries, therefore, each node is made up of more than one shape. The main program of QGERTNET loads the symbols from LOGIC.CHARSET into memory by calling procedure GETSHAPES. When a drawing command is selected, procedure MYPLOT combines the shapes to draw the desired symbol. Below is an illustration of the shapes NODEL and NODER used to create a regular node.

```
         NODEL                   NODER
         XXXXXXXXXXXX   XXXXXXXXXXXXX
      XXX         X             X    XXX
    XXX           X             X      XXX
   XX             X             X        XX
  XX              X             X         XX
 XX               X             X          XX
 X                X             X           X
XX                X             X           XX
X                 X             X            X
X                 X             X            X
XXXXXXXXXXXXXXXXXXXXX   XXXXXXXXX            X
X                 X             X            X
X                 X             X            X
XX                X             X           XX
 X                X             X           X
 XX               X             X          XX
  XX              X             X         XX
   XX             X             X        XX
    XXX           X             X      XXX
      XXX         X             X    XXX
         XXXXXXXXXXXX   XXXXXXXXXXXXX
```

Figure 7. Combining Shapes to Form a Regular Node.

Besides the shapes in LOGIC.CHARSET, the program utilizes Turtlegraphics' MOVETO command to draw activity lines connecting the nodes in the network by moving between two specified (x,y) coordinates. Turtlegraphics is also used to display all text within the nodes and along the activity lines on the high resolution screen,

Program QGERTNET

Once the shapes had been saved, the program QGERTNET was written. Depending on the user's command selection, QGERTNET will display the network on the high resolution screen and gather information about each symbol as the network is built. Figure 8 depicts the general structural flow through QGERTNET.

The main program of QGERTNET contains the highest level of operations. The program first initializes variables and retrieves the shapes from the file LOGIC.CHARSET. Next, the tablet is readied through SETUPAD, the graphics and text screens cleared, and then, LISTMODE provides the initial prompt to the user. The program then enters a continuous loop between procedures GETXY and MYPLOT. Within the loop the program awaits an input selection from the user. Figure 9 contains the logic flow for the main program in QGERTNET.

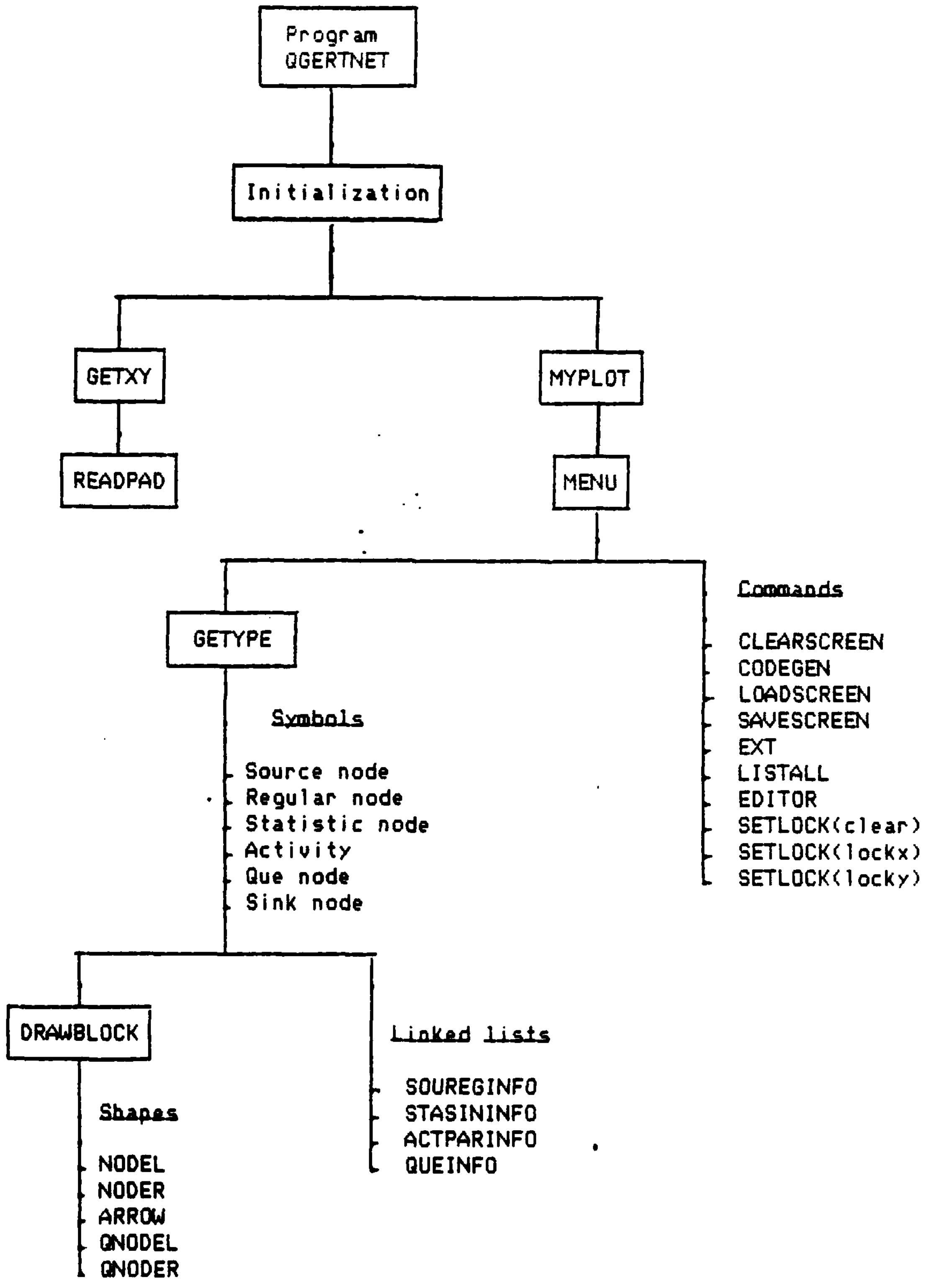


Figure 8. General Structure Diagram of QGERTNET.

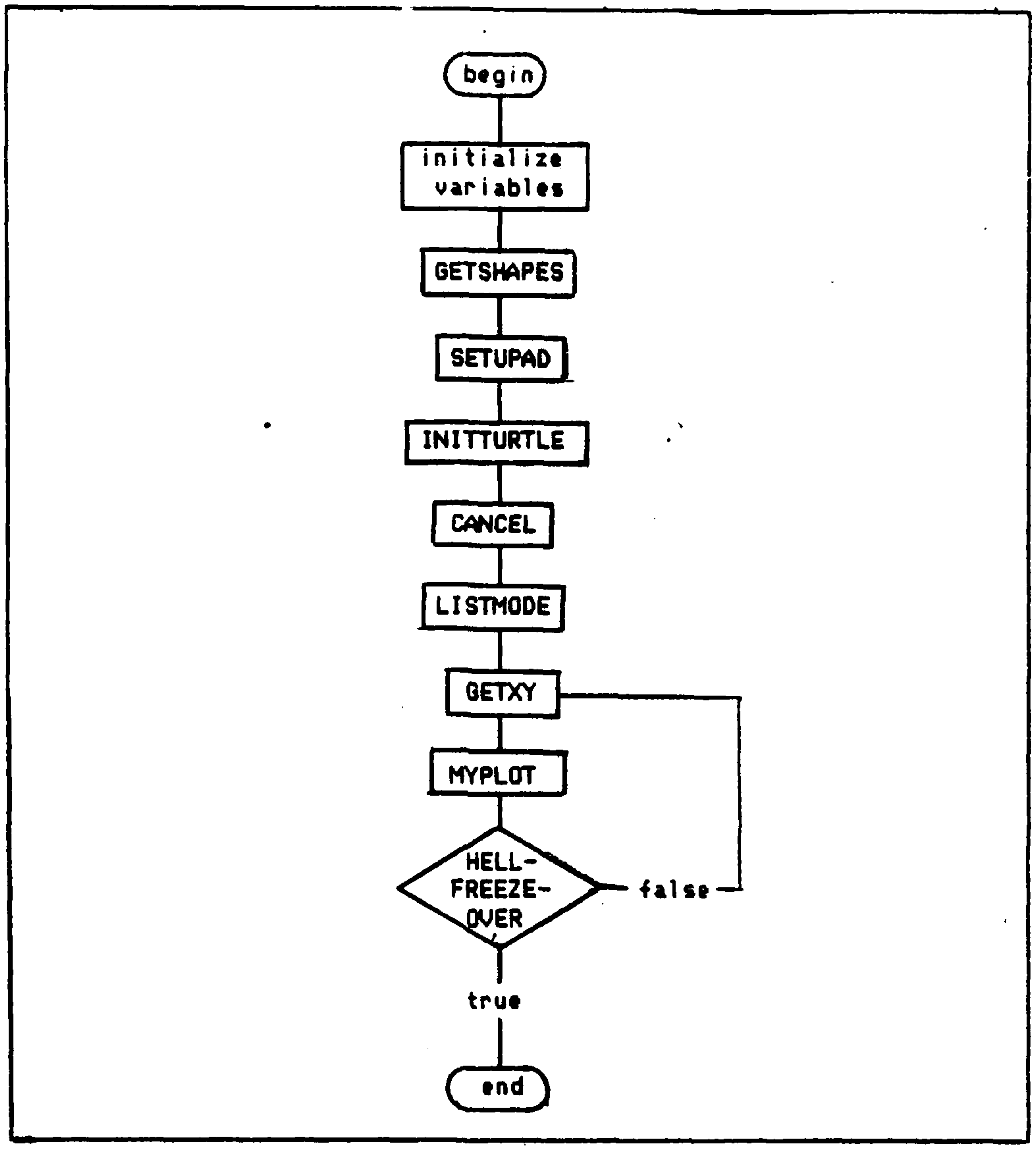


Figure 9. Logic Flow of QGERTNET's Main Program.

GETXY.

Procedure GETXY is called by QGERTNET's main program. It is used to determine the command selected, symbol to be drawn, or a symbols' placement on the high resolution screen. It first sets up memory locations for the (x,y) coordinates from the user's pen placement on the tablet.

Through procedure READPAD, the coordinates are read and stored; then, checked to determine if a valid location is selected. If the user presses the pen in the menu rows, the boolean variable VALIDXY is set false. If, on the other hand, the user, by pressing the pen in the active area, designates a position to place a symbol, VALIDXY is set true. Regardless of the coordinates saved, the program exits GETXY and enters MYPLOT. The logic flow of procedure GETXY follows:

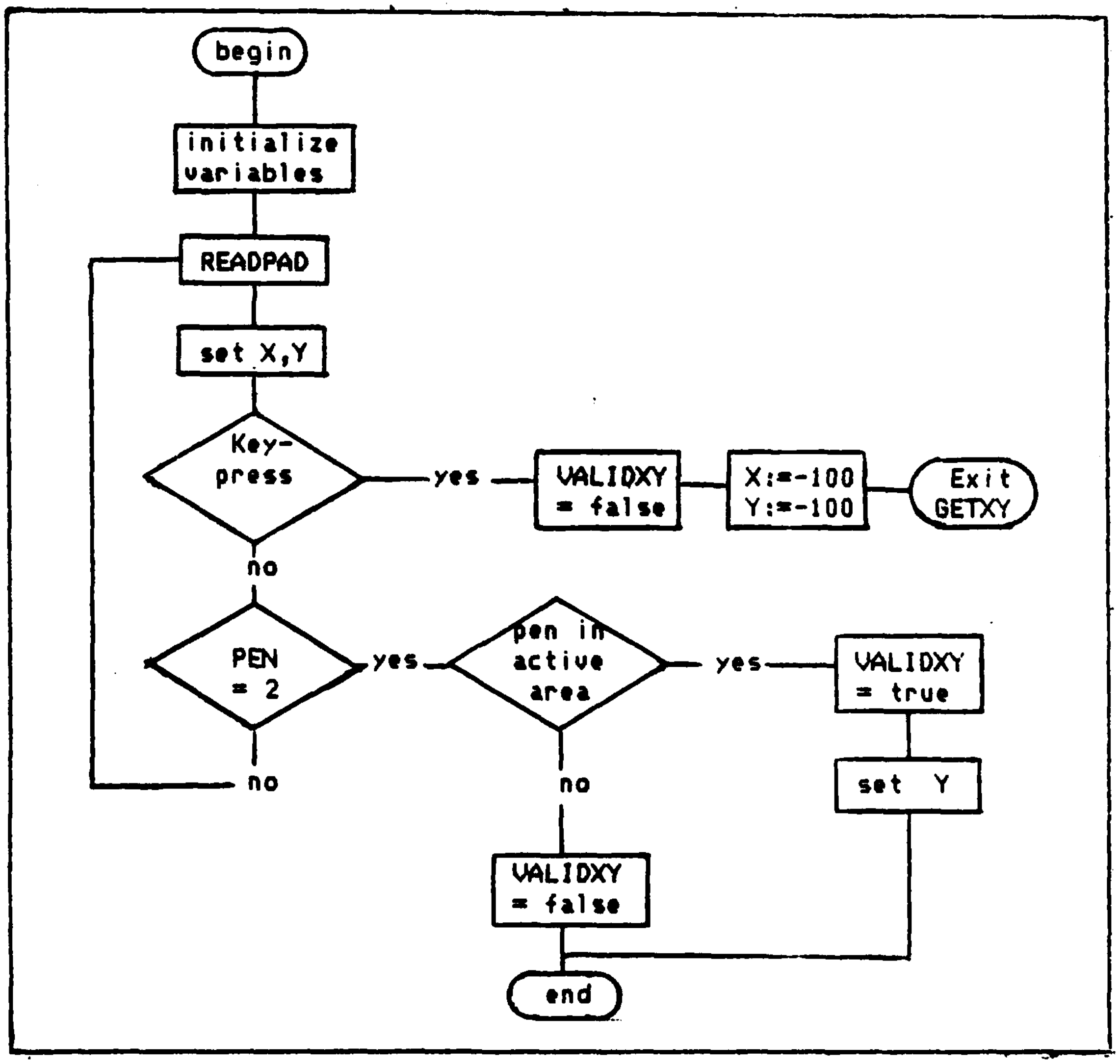


Figure 10. Logic Flow of Procedure GETXY.

MYPLOT.

In procedure MYPLOT, all the drawing and development of the linked lists is accomplished. Prior to entering MYPLOT, the boolean variable VALIDXY is set; false, if the user selected a command; or true, if the pen was pressed in the active area. If VALIDXY is false, MYPLOT calls the procedure MENU. From within MENU, the program executes the selected command or, if a drawing command was selected, returns to MYPLOT.

Having chosen a drawing command, MYPLOT draws the symbol on the high resolution screen positioned by the user's pen press in the active area of the tablet. If the pen is positioned outside the active area, VALIDXY is set false and the command terminates returning the user to MENU. When a symbol is drawn, a record of information is also generated for the symbol. As the network grows, each record is created and added to the applicable linked list by procedures SOUREGINFO for source and regular nodes, STASININFO for statistic and sink nodes, QUEINFO for queue nodes, or ACTINFO for activities and parameters. The following is the logic flow for MYPLOT.

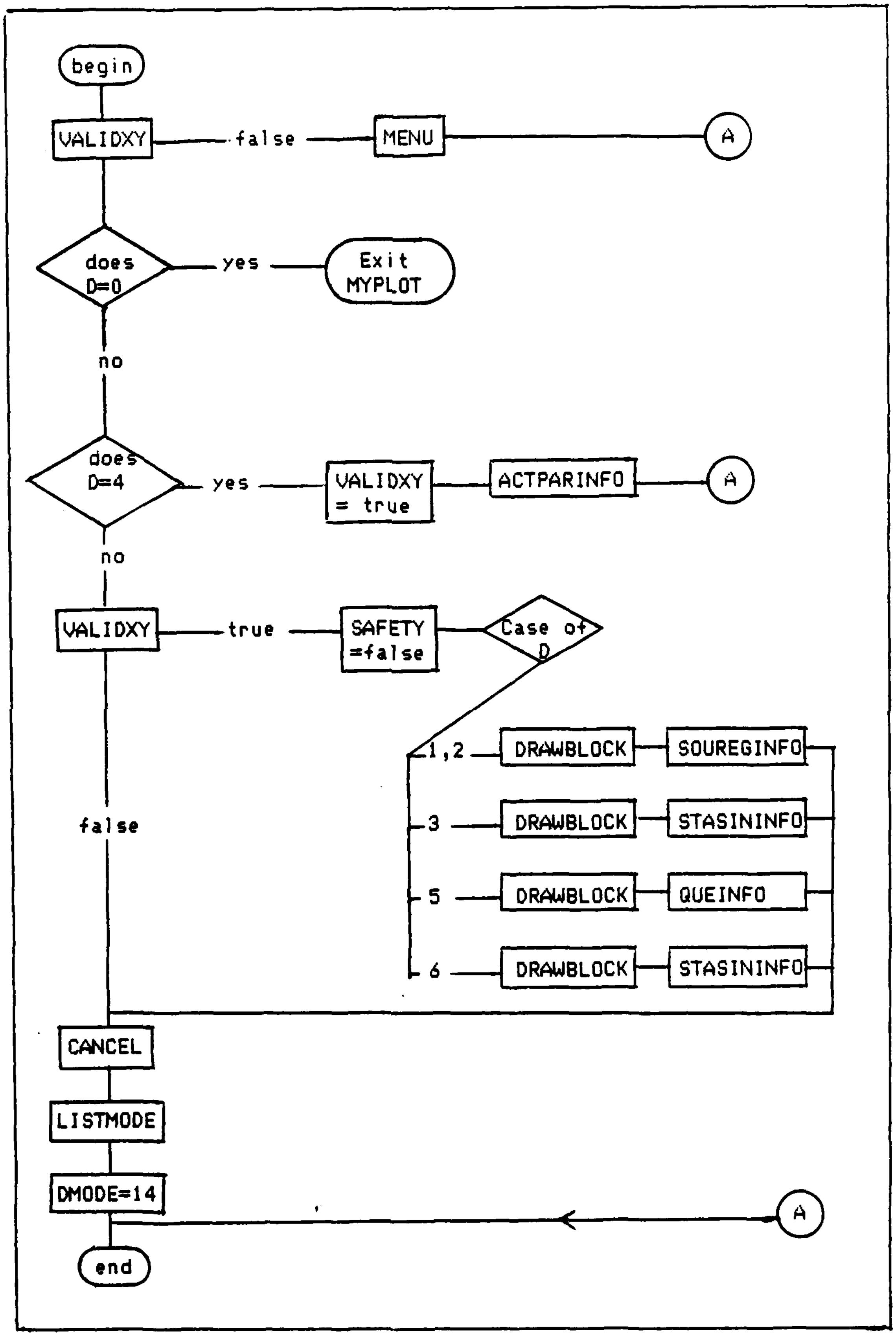


Figure 11. Logic Flow for Procedure MYPLOT.

MENU.

Upon entering MYPLOT, VALIDXY is checked. If VALIDXY is false, indicating a command was selected, procedure MENU is executed. MENU, first, determines the exact command block selected from the (x,y) coordinates passed by GETXY and transforms them into integers XPOS and YPOS. YPOS is the row of blocks and XPOS is the command block within the row. Below is a brief description of the programmed functions in the bottom row.

CLR SCR	CODE GEN	LOAD SCR	SAVE SCR	EXT	LIST	EDIT	CLR LOCK	LOCK X	LOCK Y	. . .

Figure 12. Bottom Row Menu.

CLRSCR	: clears the high resolution screen and empties the linked lists
CODEGEN	: chains the program CODEGEN to generate source code
LOADSCR	: loads the linked lists from disk into the computer memory then draws the network
SAVESCR	: saves the current linked lists to disk
EXT	: exits the program
LIST	: lists the available Q-Gert symbols
EDIT	: enters the editing procedure
CLR LOCK	: clears the x and y locks
LOCKX	: locks the x axis
LOCKY	: lock the y axis

If the row selected is either the second or third, MENU sets the integer variable D to identify the symbol, then, calls procedure GETYPE. GETYPE displays the symbol selected on the text screen employing procedure PRINTYPE. Upon MENU's termination, MYPLOT is exited and the program returns to the main program loop. GETXY is executed again so the user can position the symbol on the high resolution screen by pressing the pen in the tablet's active area. This sets VALIDXY true and re-enters MYPLOT to draw the symbol. Below are the drawing commands available to the user.

CURRENTLY UNUSED						. . .	ROW 1
SOU	REG	STAT	ACT	QUE	SIN	. . .	ROW 2

Figure 13. Rows 1 and 2 of Tablet's Menu.

SOU : draws a source node and creates a SOUREG record

REG : draws a regular node and creates a SOUREG record

STAT : draws a statistics node and creates a STASIN record

ACT : draws an activity line and creates a ACT record

QUE : draws a queue node and creates a QUE record

SIN : draws a sink node and creates a STASIN record

The logic flow for procedure MENU follows.

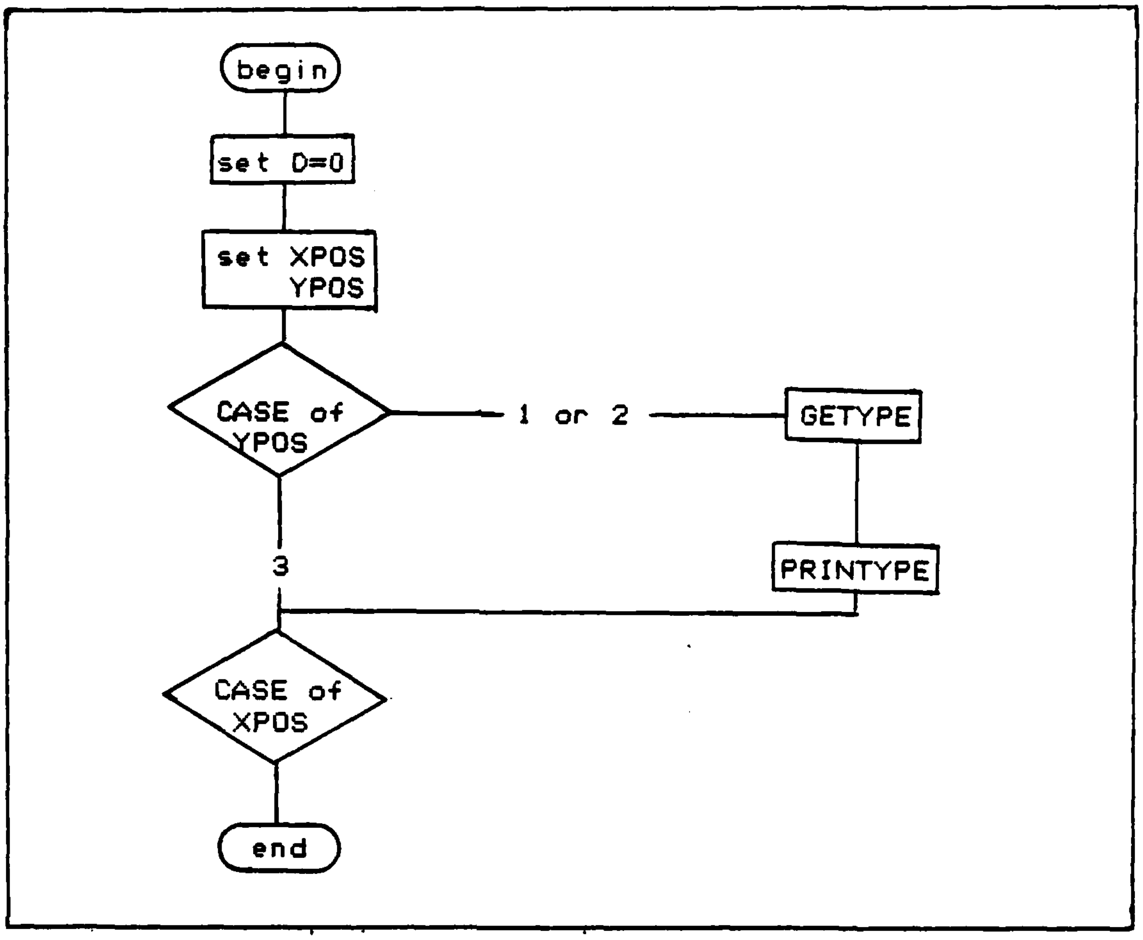


Figure 14. Logic Flow of Procedure MENU.

SOUREGINFO, STASININFO, ACTINFO, QUEINFO.

After the user selects a symbol and indicates its intended position on the high resolution screen, MYPLOT draws the symbol by combining the necessary shapes, then, calls a procedure to build a record of information about the symbol. Record building for source or regular nodes is accomplished by SOUREGINFO; for statistic and sink nodes by STASININFO; for queue nodes by QUEINFO; and for activities and parameters by ACTINFO. Regardless of the procedure, if the information is available, the program automatically fills the fields within the record else the user is asked to input the information. As the network is developed, all source and regular node records are linked together forming a linked list which grows as the number of source and regular nodes increases. This is also true for statistic and sink nodes, queue nodes, activities, and parameter linked lists. Records within the linked lists can later be saved to disk, edited or deleted.

In procedures SOUREGINFO, STASININFO, and QUEINFO the fields device (D), xlocation (X), ylocation (Y), and tipe ('Sou', 'Reg', 'Sta', 'Sin', 'Que') are all automatically filled with information obtained from MENU, GETYPE, and GETXY. User solicited information fills the remaining fields, such as, node number, initial number of transactions to release the node, subsequent number of transactions to release the node, capacity of the queue, marking, type of statistics to gather, and comment, if any. The fields pertaining to intermediate or advanced Q-Gert concepts are set to the default values, for instance, branching ('D') and attribute choice criterion ('L'). Figure 15 lists the fields applicable to each Q-Gert symbol available in this program.

In the procedure ACTINFO, both the information for activity and parameter records is obtained. These features were combined since the designation of a parameter set in an activity record necessitates the generation of a corresponding parameter record. If the activity's service time distribution is constant or the user designates a duplicate parameter set, ACTINFO will not request information to

Fields*

1	2	3	4	5	6	7	8	9	10
REG or SOU	Node number	Initial number to release [1]	Subsequent number to release [∞]	Branching* (D,P,F,A) [D]	Marking (M) [M if SOU, no M if REG]	Choice criterion (F,L,S,B) [L] / Attribute [M]			
SIN or STA	Node number/ label	Initial number to release [1]	Subsequent number to release [∞]	Branching* (D,P,F,A) [D]	Statistics desired (F,A,B,I,D) [F]	Upper limit of first cell [N]	Width of histogram cell [N]	Choice criterion (F,L,S,B) [L]/ Attribute [M]	
QUE	Node number/ label	Initial number in queue [0]	Capacity of Q-node [∞]	Branching (D,P) [D]	Ranking (F,L,S,B) [F]/ Attribute [M]	Block or node number for balkers (B) [balkers destroyed]	Upper limit of first cell [N]	Width of histogram cell [N]	Following* S-nodes or match nodes or allocate nodes
PAR	Parameter set number	Parameter 1 [0]	Parameter 2 [-10^{20}]	Parameter 3 [10^{20}]	Parameter 4 [0]	Stream* number [10]			
ACT	Start node	End node	Distribution or function type [CO]	Parameter set or constant [0.0]	Activity number/ label	Number of parallel servers [1]	Probability* or attribute number or order [.5]	Condition code* [Ni.R] i=start node	

* field set to default
[] Default values

[14]

Figure 15. Available Q-GERT Symbol Input.

Distribution and Function Types		Parameter Values			
Code	Key	1	2	3	4
AT	Attribute	-	-	-	-
BE	Beta	μ	a	b	σ
BP	Beta PERT	m	a	b	-
CO	Constant	μ	-	-	-
ER	Erlang	μ/k	a	b	k
EX	Exponential	μ	a	b	-
GA	Gamma	μ	a	b	σ
IN	Incremental	-	-	-	-
LO	Lognormal	μ	a	b	σ
NO	Normal	μ	a	b	σ
PO	Poisson	μ-a	a	b	-
TR	Triangular	m	a	b	-
UF	User Function	-	-	-	-
UN	Uniform	-	a	b	-

$\mu \rightarrow$ mean; $\sigma \rightarrow$ standard deviation;
m $\rightarrow$ mode; a $\rightarrow$ minimum or optimistic time;
b $\rightarrow$ maximum or pessimistic time.

[14]

Figure 16. Distributions and Parameter Values.

generate a parameter record. To create an activity record, the program asks for the start and ending nodes. ACTINFO searches the node linked lists by calling SEARCH to locate the two records and get the (x,y) coordinates from the nodes xlocation and ylocation fields. These coordinates are placed in the activity record fields SX and SY for the start node and EX and EY for the ending node. Turtlegraphics' MOVETO command draws a line on the high resolution screen connecting the two sets of coordinates. The user must also answer the prompts to fill the fields pertaining to service distribution, parameter set number, activity number, number of servers, and comment. As indicated previously, if a distribution other than a constant or redundant parameter set is specified, ACTINFO will ask the user for the four parameters for the specified distribution. Figure 16 depicts the distributions available in Q-Gert and the parameter values that must be specified.

As the information is entered into the symbols' record fields, the same information is written to the high resolution screen within the symbol or centered along an activity. Procedure WRITEONSCREEN uses the (x,y) coordinates of a node as a bases for placing the text in the proper locations. Activity information is displayed by centering the text in both the x and y directions along the activity line connecting the two nodes.

Below is a depiction of a regular node and the placement of text within it.

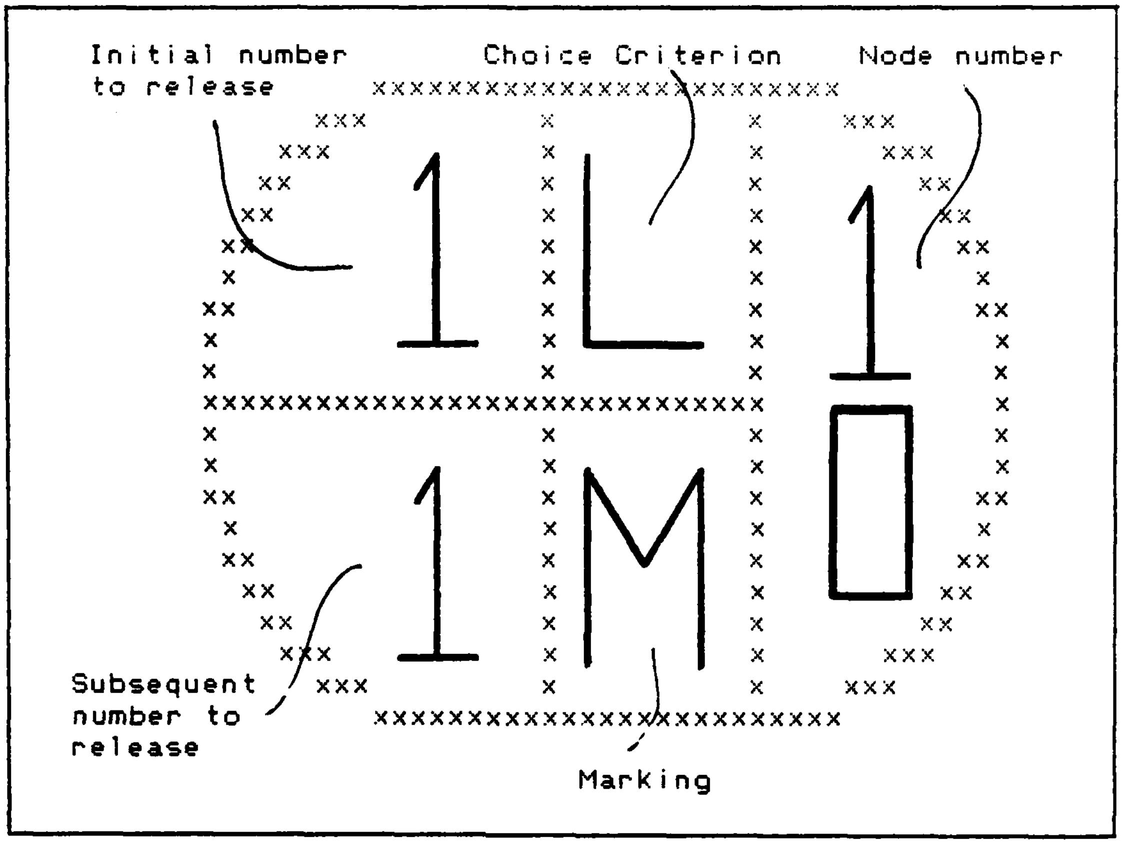


Figure 17. *Text Location Within a Regular Node.*

The following are the logic flows for procedures SOUREGINFO and ACTINFO which represent two variations of the INFO procedures.

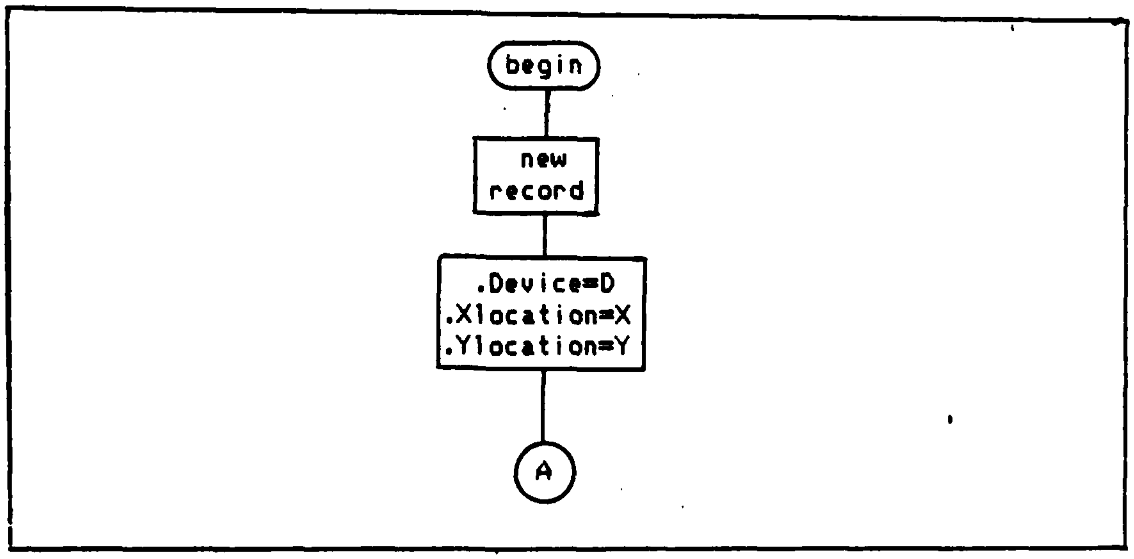


Figure 18. Logic Flow for Procedure SOUREGINFO.

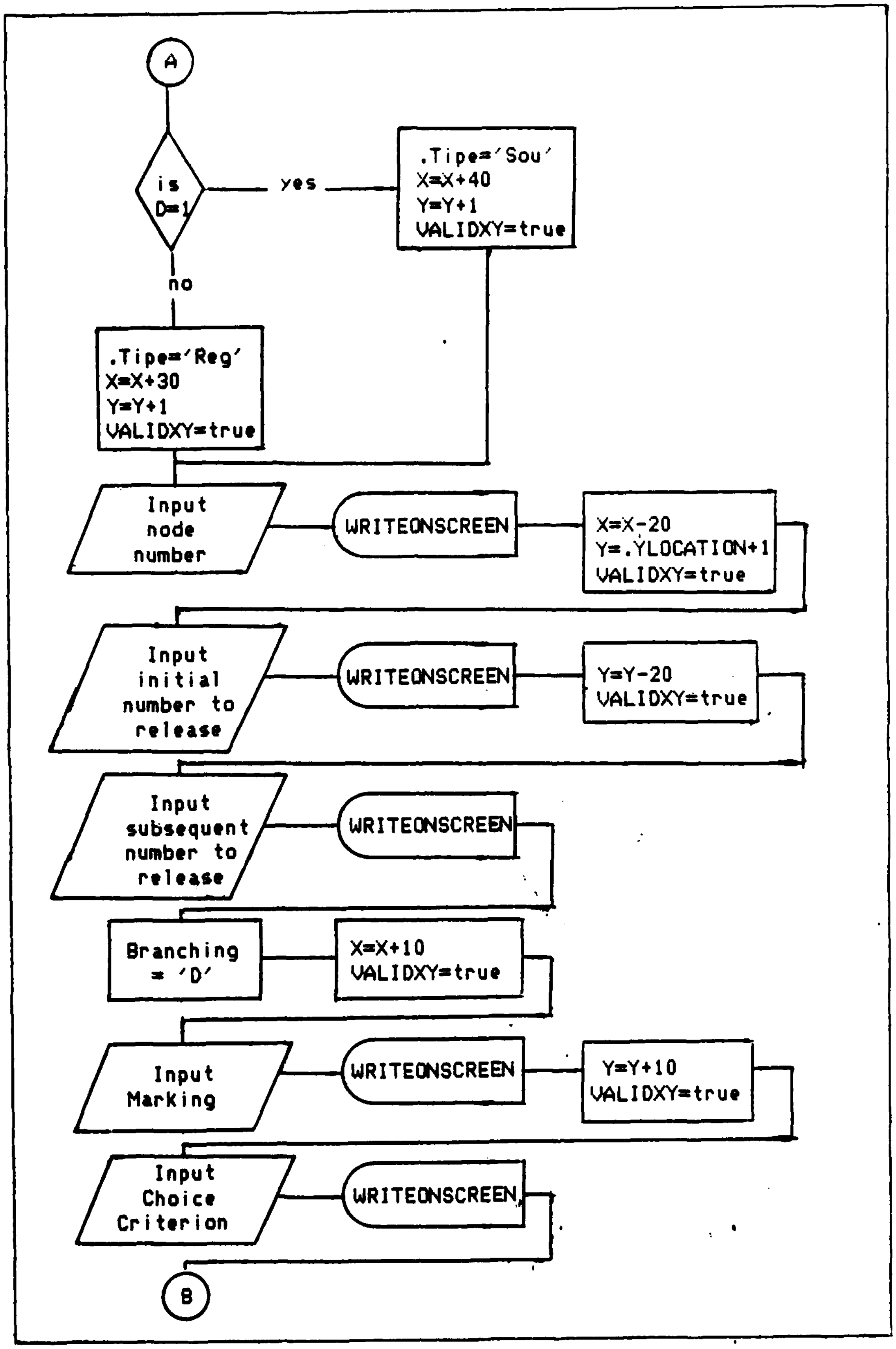


Figure 18. Logic Flow for Procedure SOUREGINFO (con't).

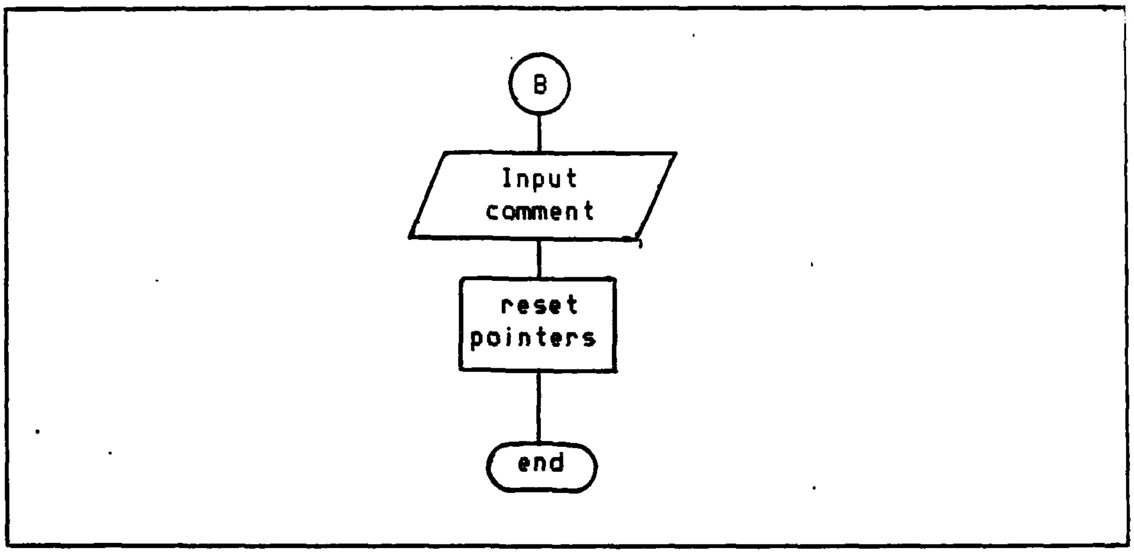


Figure 18. Logic Flow for Procedure SOUREGINFO (con't).

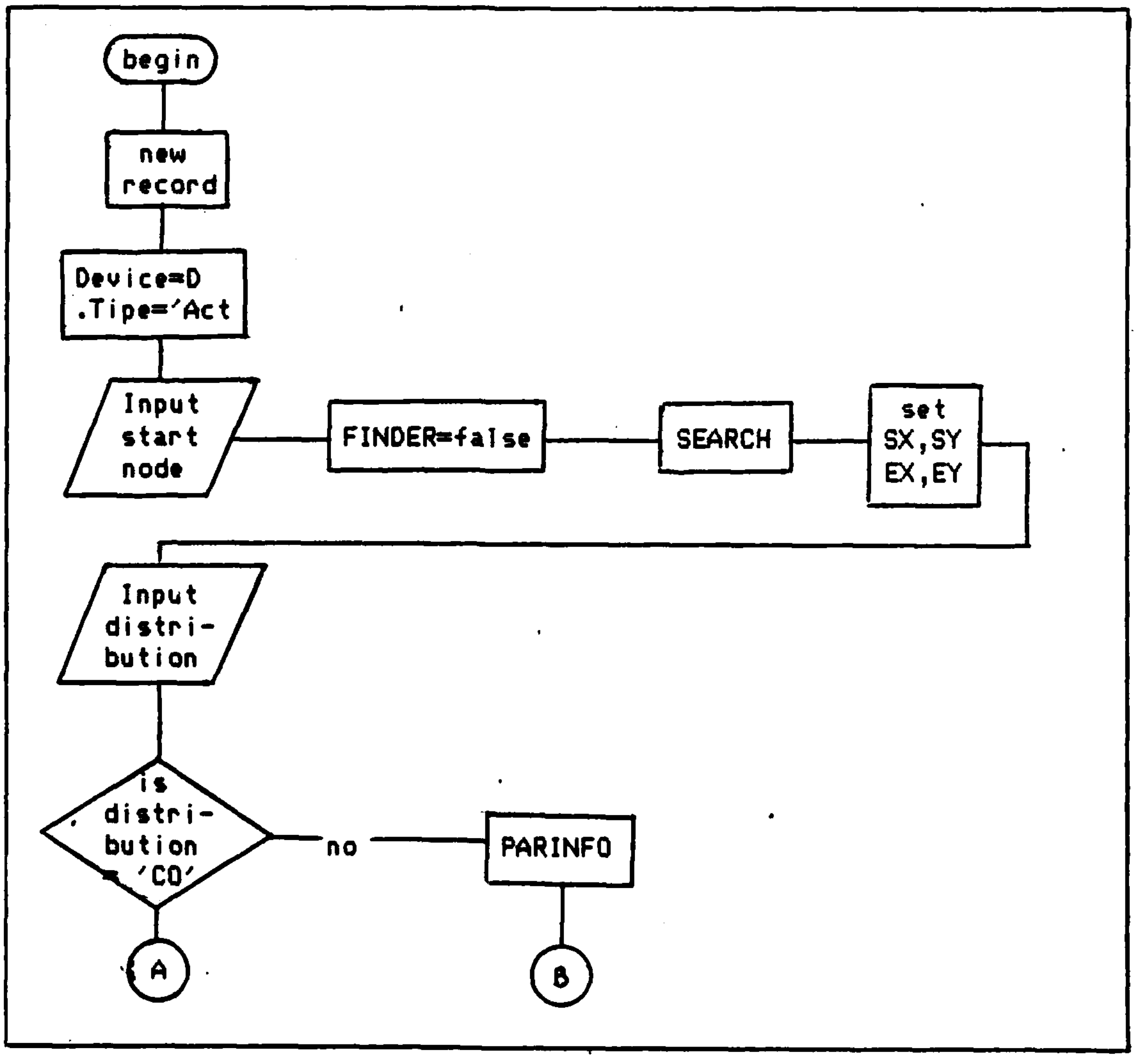


Figure 19. Logic Flow for Procedure ACTINFO.

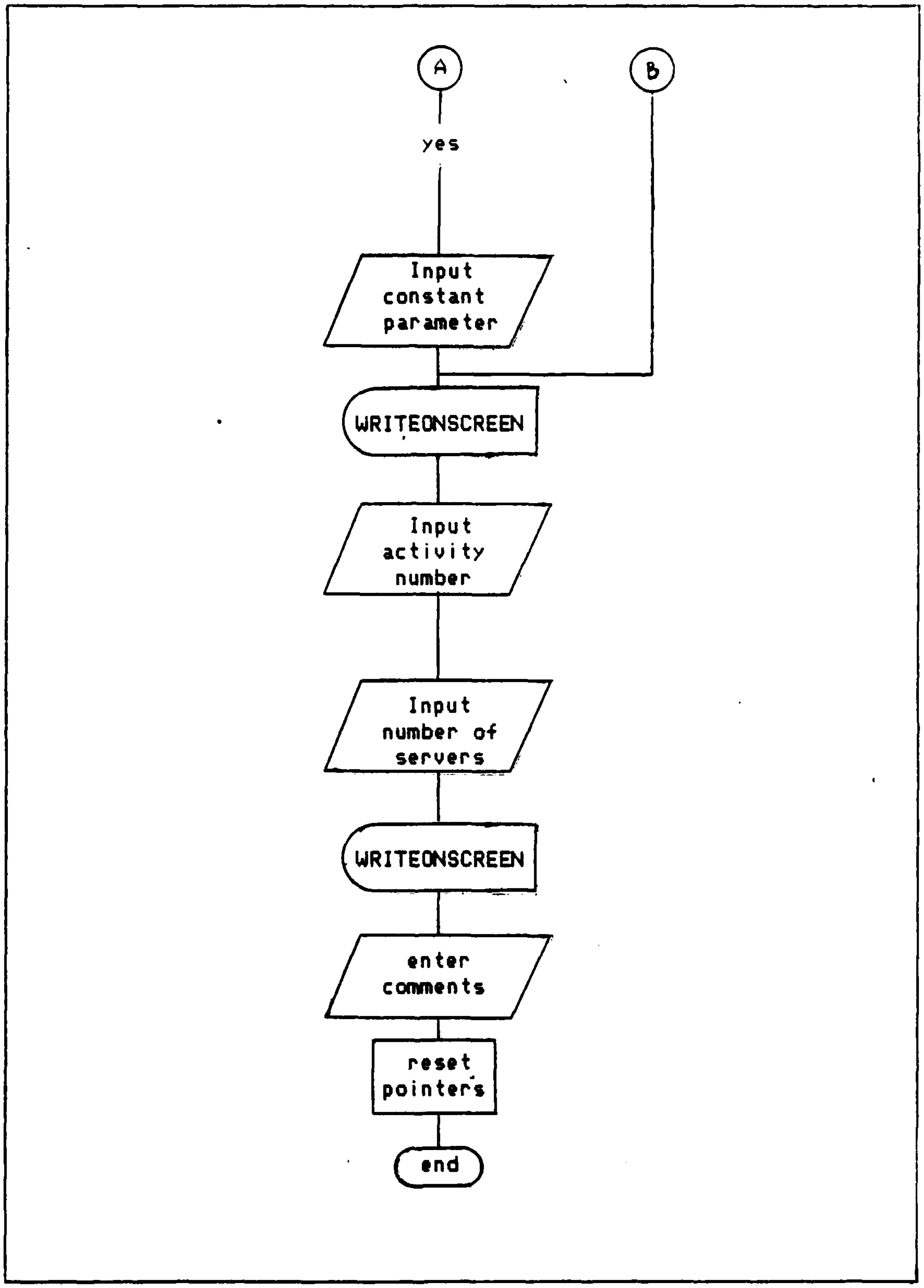


Figure 19. Logic Flow for Procedure ACTINFO (con't).

SAVESCREEN.

In order to save a network, procedure SAVESCREEN was developed. It may be called directly by the user from the command row, or indirectly by updating changes made during editing. When saving a network, only the records within the link lists are retained. The user must indicate a filename for the network which becomes the prefix to the files which will contain the records of each linked list.

SAVESCREEN is a single command procedure that calls procedure OUTDATA and passes record type parameters representing the different types of records contained in the link lists. OUTDATA opens a file for each type of record created and puts each record from the respective linked list into the file before closing. A file is built for source and regular nodes, statistic and sink nodes, activities and parameters. A file is built even if no records of that type exist. Also in OUTDATA, the boolean variable SAFETY is set true indicating that the network has been saved which facilitates exiting the program. Below is the logic flow of procedure OUTDATA.

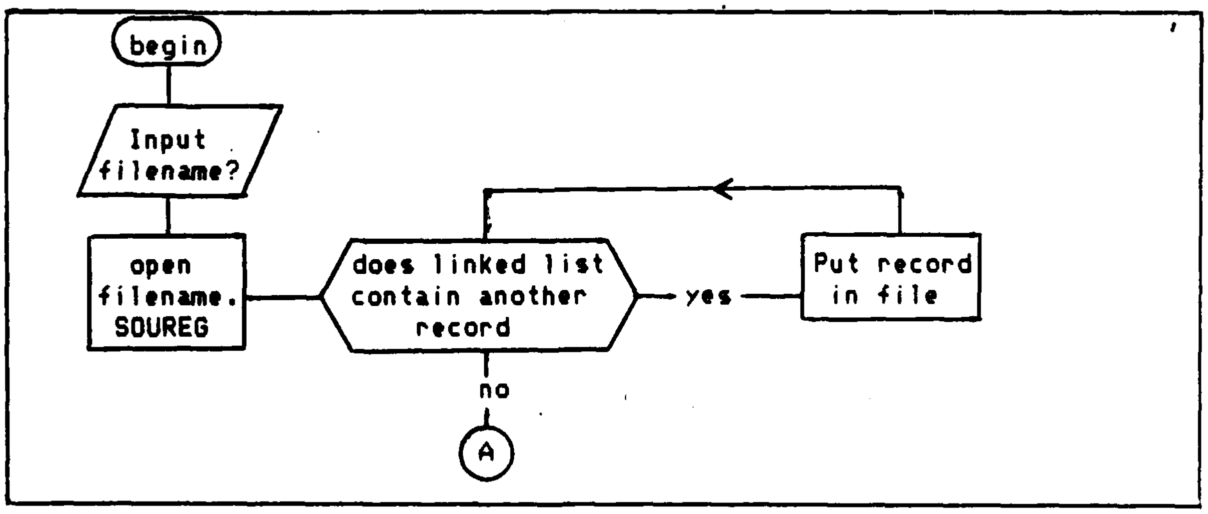


Figure 20. Logic Flow for Procedure OUTDATA.

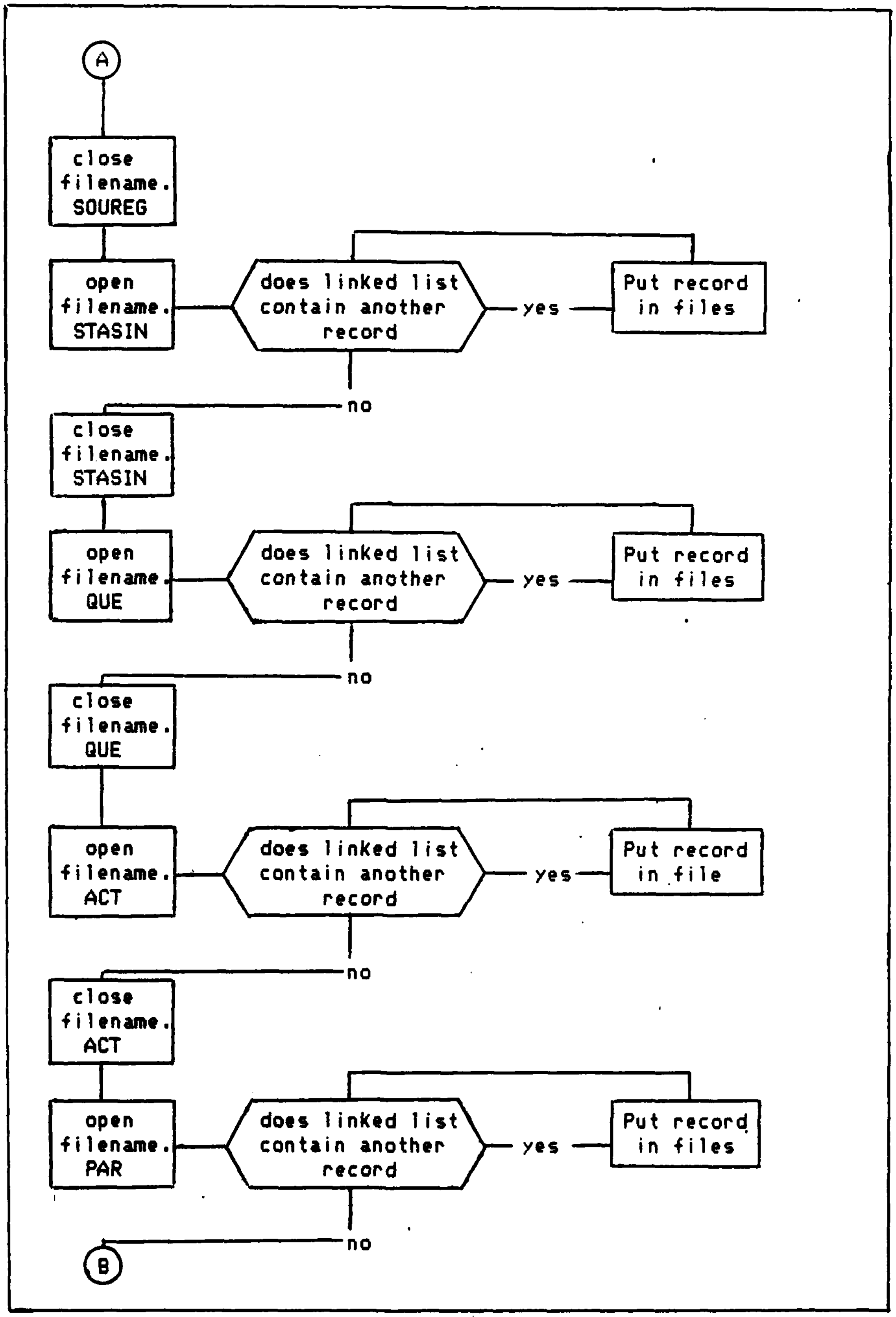


Figure 20. Logic Flow for Procedure OUTDATA (con't).

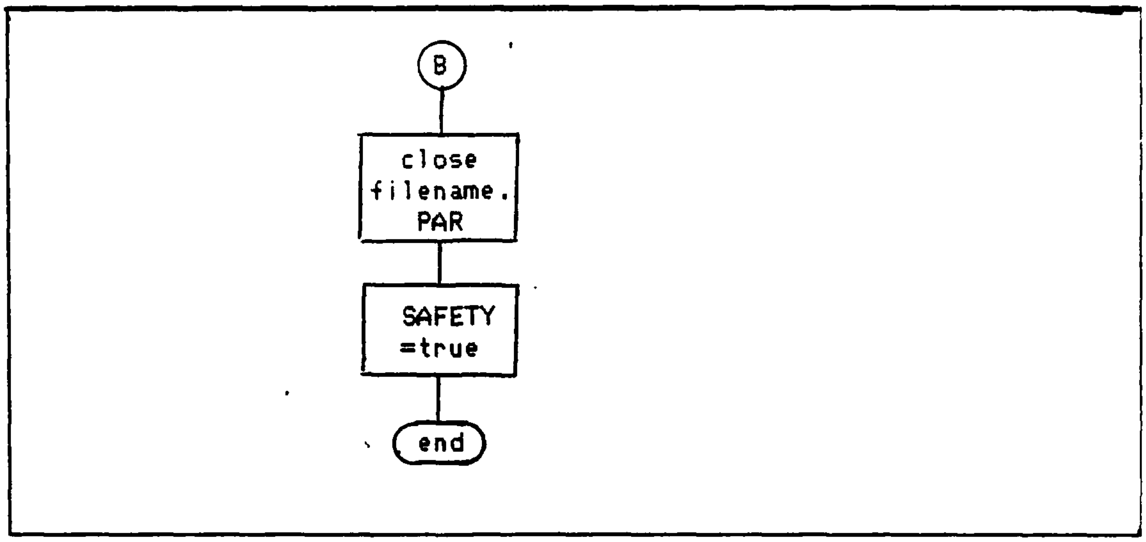


Figure 20. Logic Flow for Procedure OUTDATA (con't).

LOADSCREEN.

Procedure LOADSCREEN can be called from MENU or program CODEGEN. If the command to load a network is selected from the tablet's bottom row, then MENU calls LOADSCREEN directly. Program CODEGEN calls LOADSCREEN when the user wants to generate source code on a previously saved network. LOADSCREEN's only command calls procedure INDATA. The parameters passed to procedure INDATA are the files of records created when the network was saved.

INDATA asks the user to input the filename of the network to load. The procedure checks to see if the files exists, and if so, opens the files prefixed by the filename given. The records within each file are read, then either procedure GRAFNODE or GRAFACTIVITY are called to draw the corresponding symbol. GRAFNODE accesses each node record for the the (x,y) coordinates to position and draw the symbol on the high resolution screen. GRAFACTIVITY draws the activity line accessing the start and end (x,y) coordinate fields previously stored within the activity records. The text associated with the symbols is retrieved from the particular fields within each record and written to the high resolution screen using procedure WRITEONSCREEN.

GRAFNODE and GRAFACTIVITY follow a similar logic flow to MYPLOT for drawing the symbols, and, the logic flow of SOUREGINFO, STASININFO, and ACTINFO for writing the text to the screen except the inputs are obtained from the records rather than the user. The logic flow of INDATA follows:

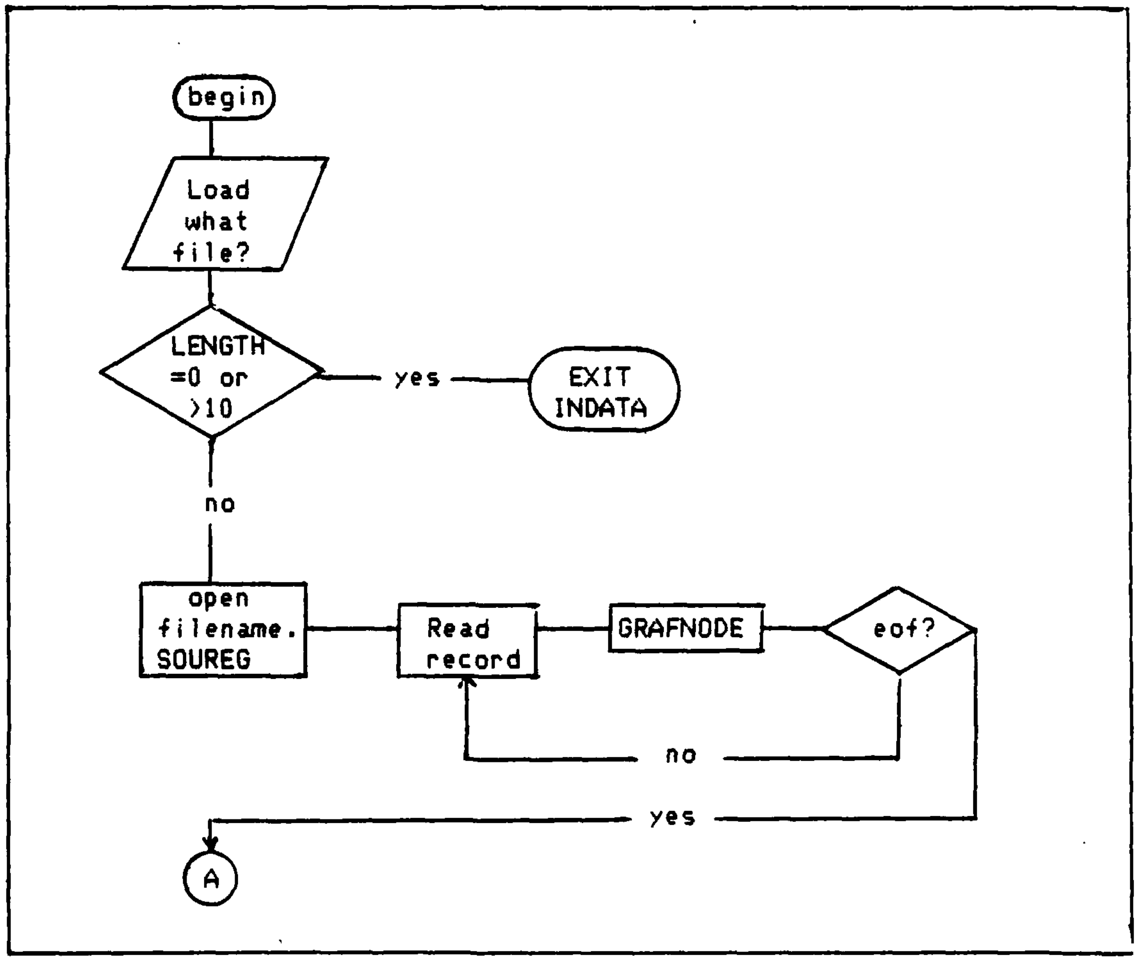


Figure 21. Logic Flow for Procedure INDATA.

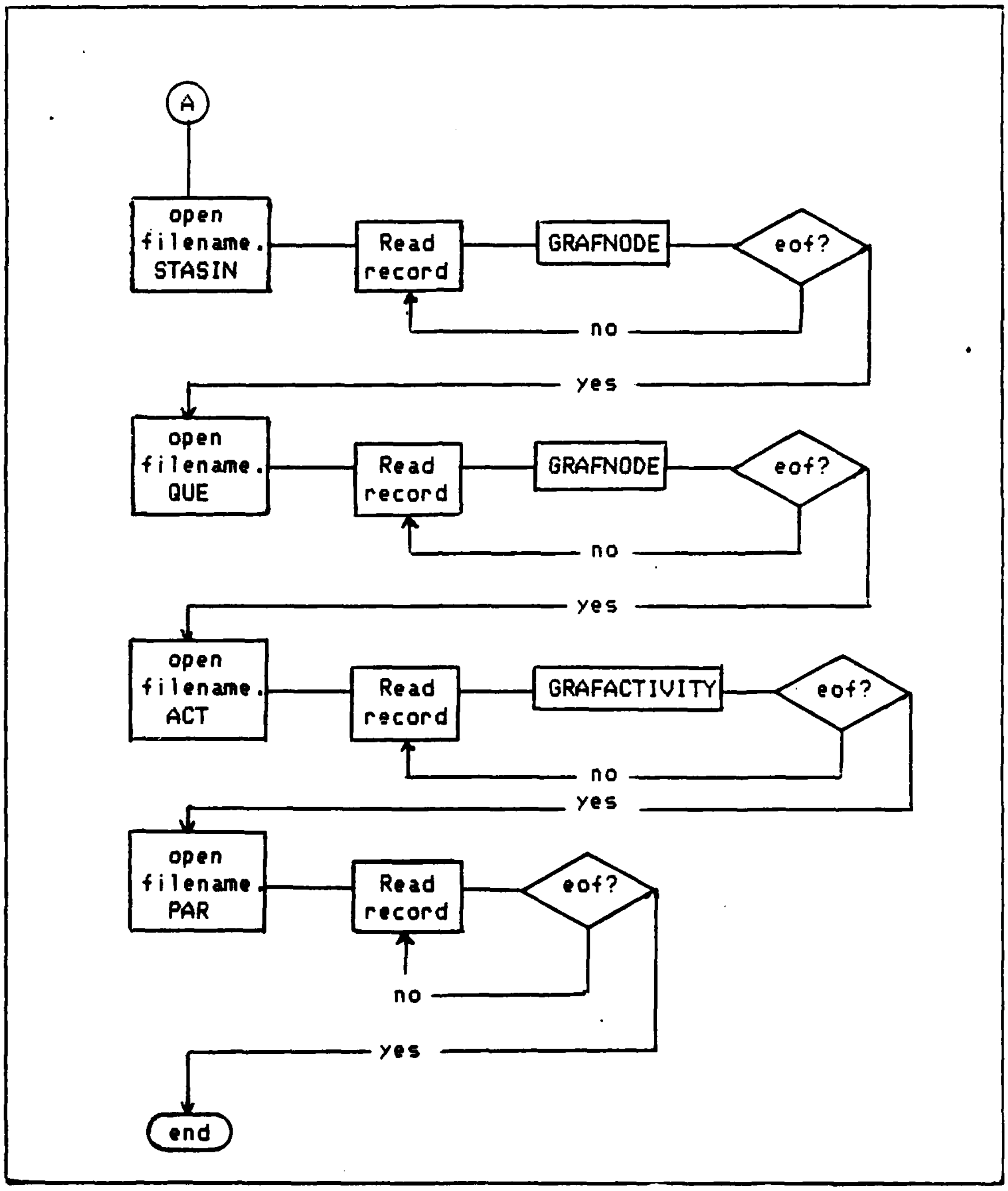


Figure 21. Logic Flow for Procedure INDATA (con't).

EDITOR.

The capability to edit a previously developed network is essential to an analyst. The ability to modify the network display and source code to test alternatives or change parameters allows sensitivity analysis of the system to be accomplished quickly. The user can save the original model then modify it and generate the new source code needed to evaluate the changes.

The editing function is selected from the tablet's command row. The current network can be edited directly or a previously saved network can be loaded for editing. Upon selecting the edit command, procedure MENU invokes procedure EDITOR. After identifying the network to edit through prompts, the EDITOR calls procedure LISTER to display the available options shown below.

A) Quit the editor
B) Source node
C) Regular node
D) Statistics node
E) Activity
F) Sink node
G) Parameter card
H) Update - to same filename

Selecting 'B' thru 'H' will call procedure OTHERWISE, while selecting 'A' will call SAVESCREEN to save the changes under the current filename then exit EDITOR. Based on the user's choice of options, procedure OTHERWISE prompts the user to identify the particular record in which the change will be made. Associated with each type of record is a FIND, GOT and CHNG procedure. Depending on the option selected, either FINDSOUREG, FINDSTASIN, FINDQUE, FINDACT, or FINDPAR search their respective linked list to find the record and display its contents on the text screen through the respective GOT procedures.

The FIND procedures first ask if the record will be deleted. If 'yes', the record is eliminated from the respective linked list by directing the pointers around the specified record. When the linked list is saved, the deleted record is not read into the file. If the delete response is 'no', the respective CHNG procedure is invoked. The CHNG procedures present a list of options to the user regarding the fields that can be changed. The options available, however, only include fields that will not effect the layout of the network. For instance, the user cannot change the position in the network of a node, he must first delete the old node and redraw it in the new position. He must also delete the activities that may have been associated with the node since the SX, SY, EX, and EY fields may no longer coincide with the new position of the node. Once a change has been made, selecting option 'H' will update the network by saving the records (SAVESCREEN) under a specified filename and displaying the updated network on the high resolution screen (LOADSCREEN). The logic flow of procedure EDITOR follows.

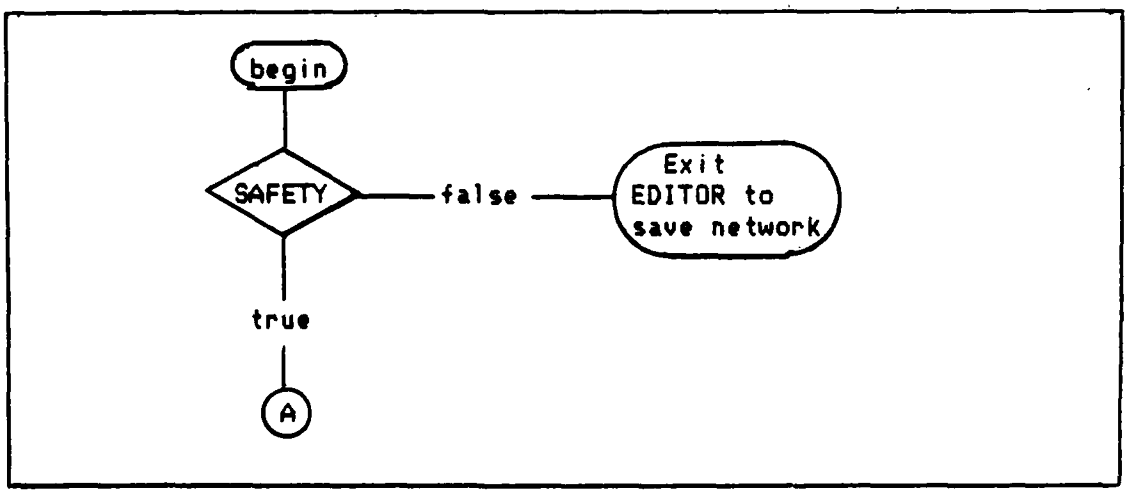


Figure 22. Logic Flow for Procedure EDITOR.

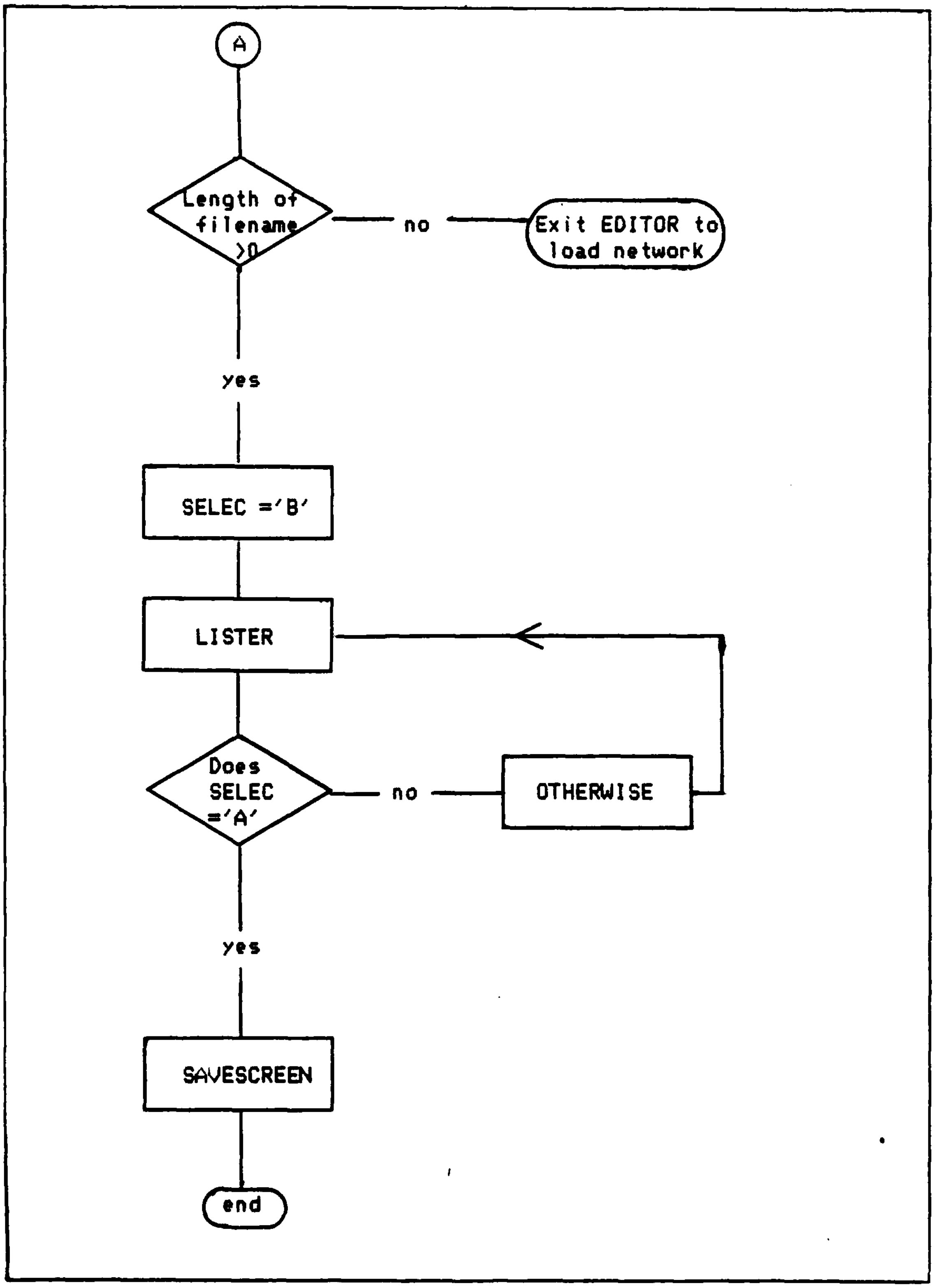


Figure 22. Logic Flow for Procedure EDITOR (con't).

Program CODEGEN

Program CODEGEN culminates the development of the Q-Gert network. This program uses the records developed in QGERTNET to produce a text file containing the Q-Gert source code. CODEGEN is a separate program because of it's size and and the timing that the user would invoke the program. The user normally generates source code upon completion of a network, therefore, CODEGEN can be called from QGERTNET to provide uninterrupted source code generation. The user can also enter CODEGEN as his first command selection; whereby, a previously saved network is loaded into memory and the source code generated.

CODEGEN is chained to QGERTNET by using the procedure call SETCHAIN(NEXTFILE) in MENU. Since CODEGEN is a separate program, QGERTNET must be exited before invoking it. As soon as QGERTNET is terminated, the operating system will execute the file whose name is the value of NEXTFILE. NEXTFILE in this case is the program CODEGEN.

To generate the source code, the network's record files must first be reloaded into memory using LOADSCREEN which also redraws the network on the high resolution screen. Following the initial setup, a series of administration questions is asked regarding the network in procedure GENERALCARD. Similar to generating the symbols' records, any data that can be obtained from previous inputs is automatically inserted into the general card. For instance, procedure STATSEARCH counts the statistic and sink nodes in the network. After the general card is completed, procedures SOURCESEARCH, ACTIVSEARCH, and PARSEARCH are sequentially called to access the linked lists' information to create the rest of the input cards representing the Q-Gert network.

Each record type contains all the information necessary for generating its corresponding Q-Gert source code. The SEARCH procedures step through their respective link lists building a string variable containing the complete source code for each node, activity or parameter. The order in which the strings enter the text file is important. The source node strings must come first; the start node string of an activity must precede the corresponding activity string; and, sink nodes must come after all other nodes and activities. The parameters strings can be placed either before or after the sink node strings. With these restrictions in mind, the source node strings are generated first, followed by all regular node strings, then statistic node strings. Next, all activity strings are generated followed by parameter strings, and lastly, the sink node strings. Figure 23 depicts the logic flow for CODEGEN's main program.

As a cross check, the source code is presented on the text screen. When the CODEGEN is finished, QGERTNET is re-executed using the SETCHAIN procedure allowing the user to begin work on a new network or modify an existing one.

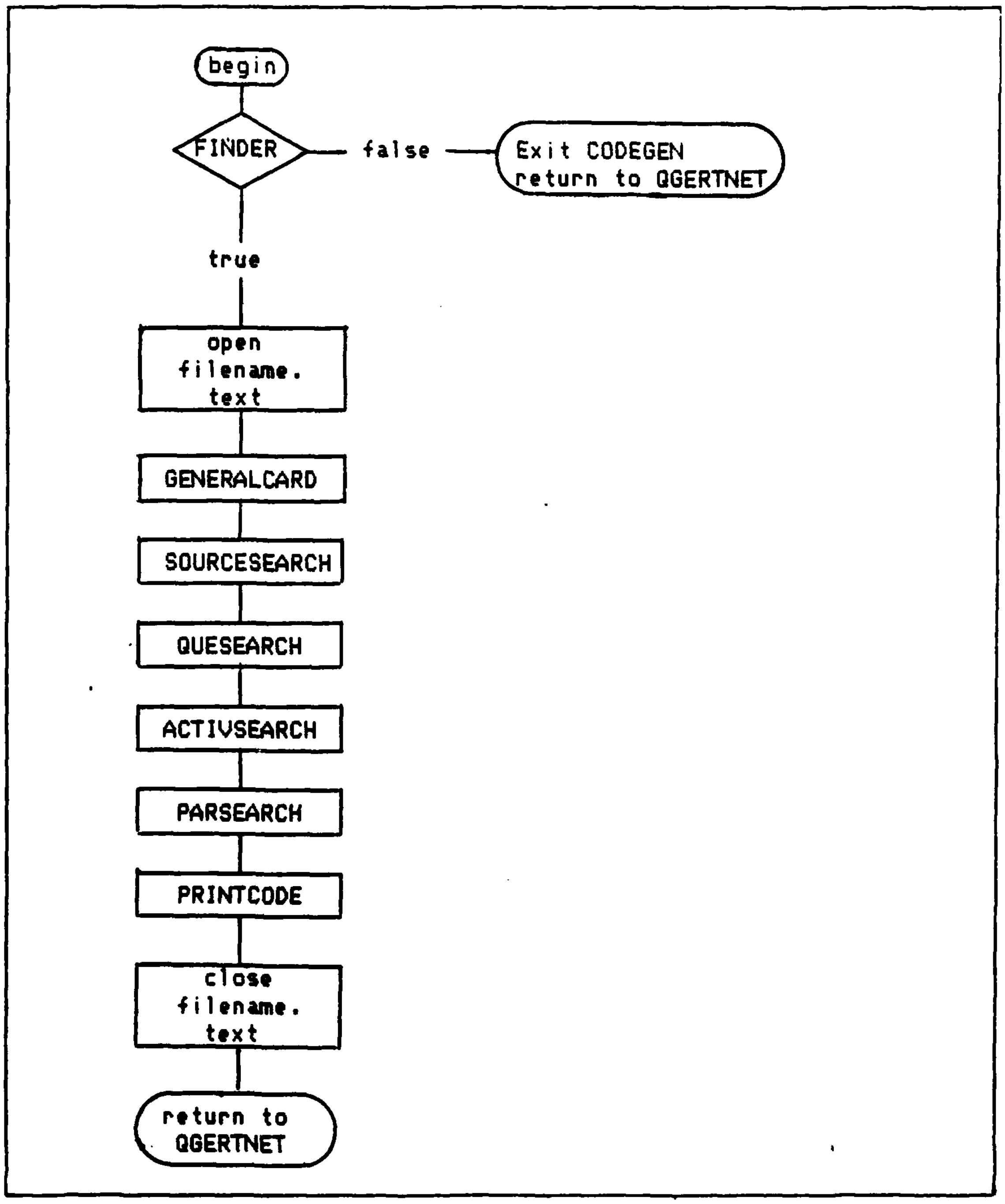


Figure 23. Logic Flow For CODEGEN's Main Program.

Units

One of the major reasons for selecting Apple Pascal as the programming language for this project was units. Units allow a large program to be divided into smaller parts which are entered into the computer memory as they are needed. This is a similar process to overlaying found in FORTRAN. This capability allows a compiled computer program to be several times larger than the available computer memory allowing increased processing and display capabilities. In addition, the variables common to all procedures are immediately available, which is not the case in other programming languages, such as BASIC.

Related procedures were combined into units. When QGERTNET or CODEGEN calls a procedure within a unit, the entire unit enters memory. Using the compiler's 'noload' option (*N+*), the unit stays in memory only as long as a procedure within it is being acted upon. The units were built so that procedures that called other procedures were grouped together to minimize the memory needed to execute the program. Below is a listing of the procedures comprising each unit.

Unit GLOBAL	Unit WRITE	Unit NODEINFO
all global variables	Writeonscreen Getshapes	Soureginfo Stasininfor Queinfo

Figure 24. Listing of the Procedures Comprising Each Program Unit

Unit INN	Unit OUT	Unit ACTPAR
Grafnodes	Outdata	Parinfo
Grafactivity	Savescreen	Actinfo
Indata		Search
Loadscreen		
Unit EDITSR	Unit EDITSS	Unit EDITQUE
Chngsoureg	Chngstasin	Chngque
Gotsoureg	Gotstasin	Gotque
Findsoureg	Findsoureg	Findque
Unit EDITACT	Unit EDITPAR	Unit EDIT
Chngact	Chngpar	Otherwise
Gotact	Gotpar	Lister
Findact	Findpar	Editor
	Unit LOADER	
	Indata	
	Loadscreen	

Figure 24. Listing of the Procedures Comprising Each Program Unit (con't).

IV. Program Capabilities and Limitations

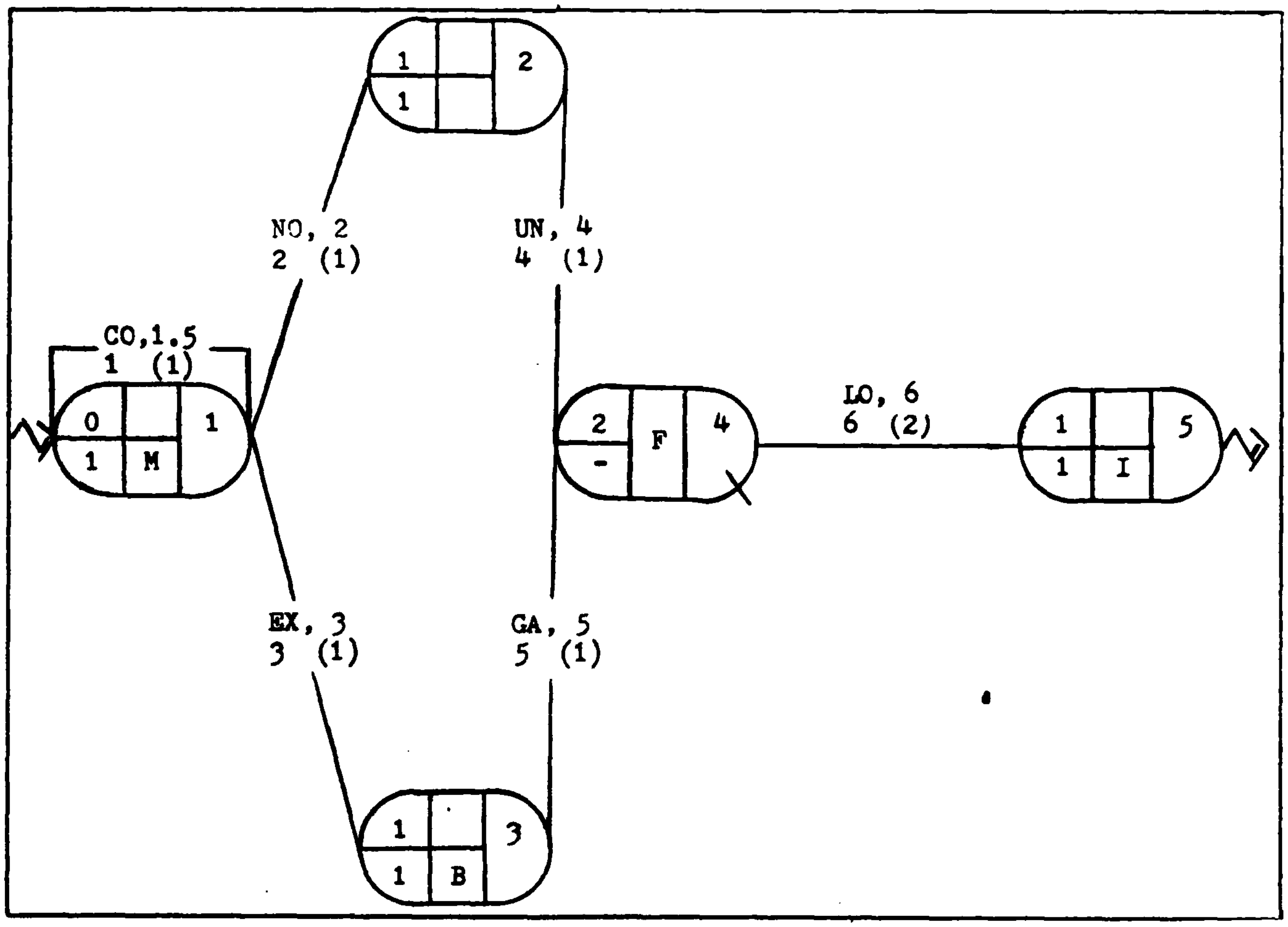
This program will enable the analyst to create a limited network using the basic concepts of Q-Gert. Available to him are source nodes, regular nodes, statistic nodes, queue nodes, sink nodes, activities, and parameters. These symbols can be used in any manner subject to the Q-Gert Analysis program limitations and the limitations of this program. Figure 24 is an example of a network display and the associated source code that can be created.

Some major limitations for the Q-Gert Analysis program that could effect the size of a network are as follows [5]:

1) Maximum number of nodes in the network is 100.
2) Maximum number of source nodes is 20.
3) Maximum number of queue nodes and select nodes is 50.
4) Maximum number of servers for an activity is 50.
5) Maximum number of activities in the network is 100.
6) Maximum number of concurrent transactions is 400.

Besides these limitations, this program is further restricted due to the limited graphic display capability. The user can only create small networks that will fit on the high resolution screen. A scrolling capability was not incorporated in this program; therefore, the network can not expand beyond the 40 column display. As a result, the Q-Gert Analysis program limitations above should not pose any size constraint on the network that can be built using this program.

Another limitation exists regarding the positioning of nodes relative to each other if an activity connects them. The program is capable of looping an activity where the start and end nodes are the same. However, care must be



```
GEN,USER,NET,1,1,1984,1,1,10,50,5,E*
SOU,1,0,1,D,M,L*                 SOURCE NODE
REG,2,1,1,D,,L*                  REGULAR NODE
STA,3,1,1,D,B,N,N,L*             STATISTIC NODE
QUE,4,2,,D,F,N,N*                QUE NODE
ACT,1,1,CO,1.5,1,1*              ACTIVITY #1
ACT,1,2,NO,2,2,1*                ACTIVITY #2
ACT,1,3,EX,3,3,1*                ACTIVITY #3
ACT,2,4,UN,4,4,1*                ACTIVITY #4
ACT,3,4,GA,5,5,1*                ACTIVITY #5
ACT,4,5,LO,6,6,2*                ACTIVITY #6
PAR,2,3.0,0.0,,0.8*              NORMAL DIST
PAR,3,2.5*                       EXPONENTIAL DIST
PAR,4,,1.0,3.0*                  UNIFORM DIST
PAR,5,3.0,0.0,10.0,0.5*          GAMMA DIST
PAR,6,4.0,2.0,6.0,0.2*           LOGNORMAL DIST
SIN,5,1,1,D,1,N,N,L*             SINK NODE
FINISH*
```

Figure 25. Sample Network Display and Associated Source Code.

exercised when positioning connected nodes because the activities are drawn based on straight lines connecting two points, if an ending node is positioned too far behind the start node, the line connecting the two will pass through the boundaries of both nodes.

Since the size of the network is restricted by this program, the symbol sizes were kept as small a possible commensurate with the size of the text displayed within the nodes' boundaries. The following minor limitations are further imposed on the users of this program:

1) Node numbers available are between 1-99.
2) Initial number of transactions to release a node must be between 0-9.
3) Subsequent number of transactions to release a node must be between 0-9 or infinity.
4) Infinity is displayed within a node as "-".

Since the program is oriented more to analyzing small PERT networks and straight forward queuing systems, some analysis capability that would normally be available have been set to the default values. Such as, there is no probabilistic branching nor choice criterion within the nodes for selecting transactions. These analysis capabilities are necessary for more complex simulations, but, this program's development has only incorporated the basic concepts of Q-Gert simulation.

V. Conclusions and Recommendations

All the questions raised by this research effort have been answered affirmatively. Using both an Apple microcomputer and the Apple Graphics Tablet, computer aided design techniques were used to generate a Q-Gert network. With Pascal's dynamic linked list data structure, records containing information about each symbol are created and stored on disk. The records can be accessed for editing or for generating the computer source code representing the Q-Gert network.

To use this program, the analyst selects from a menu, the symbol he wishes to draw, then, positions it in the network. Answering the questions asked by the program about the use of the symbol, fills a record representing the symbol. From the information contained in the records, the program will display the symbols making up the network on the high resolution graphics screen and, if desired, create a text file containing the computer source code. The source code can be sent to a mainframe computer as input to the Q-Gert Analysis program giving the analyst statistical inferences about the system simulated.

Presently, the Q-Gert network can consist of source, regular, statistic, queue, and sink nodes; activities; and parameters. These symbols comprise the basic concepts of Q-Gert and can be used to develop small simulation models to both analyze the system and present a graphical representation of the system to portray the relationships within the system.

A user's manual is contained in Appendix F. It gives a brief description of each command and steps through examples to help the user learn the existing capabilities of the program.

This program has made a giant stride in easing the burden of creating a Q-gert network and manually translating the graphical symbols into computer source code. But, there is still much that can be done to further this programs capabilities. The following are some suggested areas of improvement.

1. The foremost problem that needs to be solved is the network display size limitation. Presently, the network is limited to the size of the high resolution graphics screen. To increase the display capability, a scrolling function is needed, preferably in both the up/down and left/right directions. This would allow the network to expand up to the Q-Gert Analysis program limitations.

2. An analysis of the Pascal program may suggest programming refinements which may be more efficient and thereby able to free more memory to expand the programs capabilities.

3. Additional Q-Gert symbols and functions can be added following the same logic as presented in the program development section of this thesis. In general, to add another symbol, the follow-on programmer must add the following: shapes, to draw the symbol, to program INITLOGIC; linked list variables to unit GLOBAL; an INFO procedure to gather information to fill the symbol's record, associated FIND, SEARCH, and GOT editing procedures, additions to ACTINFO; and

lastly, a SEARCH procedure in program CODEGEN to generate the appropriate source code. Care must be taken when expanding the program to continue the memory conservation principles already in the program.

4. Sokol, in his electrical schematics program, had a printing routine that required saving the graphics display on the screen to a file before the display could be printed. Since our program does not save the graphics display to disk, we could not use his routine to print the network created. Therefore, a follow-on benefit of a printing function would give the analyst a hard copy of the Q-Gert network he created.

5. A final addition that would benefit the display capabilities is a better drawing procedure to loop back to a previously drawn node. Currently, the program can loop backwards only so far as the start and end node are the same. If a node is positioned too far behind the start node, the activity line will cross through the nodes boundaries degrading the presentation. Using Turtlegraphics and a geometry oriented function, curved lines could be used to connect the nodes when the activity's ending node is well behind the start node.

One final word, as the program is expanded to include select nodes, match nodes, vas cards, etc., the order in which the source code is entered into the text file will become important. This would require a more sophisticated search routine in program CODEGEN to ensure the proper order is maintained.

Appendix A
Procedures SETUPAD and READPAD

```
; assembly language linkage
; to APPLE GRAPHICS TABLET
;
; 20 Mar 80 - Dan Sokol
;
; procedure SETUPAD; external;
; procedure READPAD; external;

CFFF      .EQU    0CFFF     ; TURN OFF ALL ROMS
MSLOT     .EQU    7F8       ; ACTIVE SLOT = Cn
PADAT     .EQU    0C500     ; SLOT ADDR FOR PAD
MREAD     .EQU    0CEF9     ; READ THE PAD
CURSOUT   .EQU    0C8F0     ; XOR CURSOR AND SCALE PAD OUTPUTS
QWAIT     .EQU    0CCA1     ; MIDEAST COUNTRY WITH MUCH OIL
DEFAULT   .EQU    0CE90     ; SETUP PAD
DEF4      .EQU    0CEEA     ; SETUP PAGE AND MPAGE FOR SCREEN 1
DELAY     .EQU    80        ; DELAY FOR QWAIT (CURSOR ON)

          .PROC SETUPAD,0;

DFLT      LDA     CFFF      ; ALL ROMS OFF
          LDA     #0C5
          TAX
          STA     MSLOT
          LDA     PADAT
          LDA     PADAT     ; TURN PAD ON
          JSR     DEFAULT
          LDA     #20
          TAY
          JSR     DEF4      ; TURN SCREEN 1 & STREAM ON
          LDA     CFFF
          LDA     #0C3
          STA     MSLOT
          TAX               ; RESET SUP'R TERM
          RTS

          .PROC READPAD,0;

READIT    LDA     CFFF      ; ALL ROMS OFF
          LDA     #0C5
          TAX
          STA     MSLOT
          LDA     PADAT
          LDA     PADAT     ; PAD ON
          JSR     MREAD     ; READ PAD
          JSR     CURSOUT   ; FLASH CURSOR & SCALE X & Y
```

```
            LDA     #DELAY
            JSR     QWAIT
            JSR     CURSOUT
            LDA     CFFF
            LDA     #0C3
            STA     MSLOT      ; RESET SUP'R TERM
            TAX
            RTS
;
; ON EXIT >>> PEN UP/DOWN      -      640
;             SCALED X (HIGH BYTE) - 646
;             SCALED X (LOW  BYTE) - 645
;             SCALED Y (HIGH BYTE) - 648
;             SCALED Y (LOW  BYTE) - 647
```

Appendix B
Unit PEEKPOKE

```
(*************** PEEKPOKE **********************
* Adds the commands PEEK and POKE to Pascal.    *
* Intrinsic unit in System.Library              *
*        Dan Sokol     3 Dec 79                 *
***********************************************)
(*$S+*)
unit PEEKPOKE; intrinsic code 23 data 24;
interface
procedure POKE(var ADDR,DATA:integer);
function PEEK(var ADDR:integer):integer;
implementation
type
   PA=packed array[0..1] of 0..255;
   MAGIC=record case boolean of
      true:(INT:integer);
      false:(PTR:^PA);
   end;
var
   CHEAT:MAGIC;
procedure TEST(var DATA:integer); forward;
procedure POKE;
begin
   TEST(DATA);
   CHEAT.INT:=addr;
   CHEAT.PTR^[0]:=DATA;
end;
function PEEK;
begin
   CHEAT.INT:=ADDR;
   PEEK:=CHEAT.PTR^[0];
end;
procedure TEST;
begin
   DATA:=abs(DATA mod 256);
end;
begin
(* dummy program *)
end.
```

Appendix C
Program INITLOGIC

```
(*******************************************************
* This program creates the file 'LOGIC.CHARSET' which is *
* used by QGERTNET.  Each character is a 21 by 21 array  *
* (of boolean .. i.e. true or false .. 1 or 0) which is  *
* used to draw the QGERT symbols.                        *
*        Dan Sokol - 2 Apr 80                            *
*        Modified by Anderson and Commeford              *
*******************************************************)
program INITLOGIC;
type  SHAPE = packed array[0..20,0..20] of boolean;
var   NODEL, NODER, ARROW,
      QNODEL, QNODER        : SHAPE;
      SHAPEFILE             : file of SHAPE;
      I, J, ROW             : integer;
      BIT                   : boolean;
(**************** MAKESHAPES **************************
* Converts strings to boolean arrays.  This procedure is *
* borrowed from GRAFDEMO.TEXT on APPLE3:                 *
*       Called from : INIT1 THRU INIT5.                  *
*******************************************************)
procedure MAKESHAPES(var BITMAP:shape;ST:string);
  begin
    for J:=1 to length(ST) do
    begin
      BIT:=(ST[J]<>' ');
      BITMAP[ROW,J-1]:=BIT;
    end;
    ROW:=ROW-1;
  end;
(************* SAVESHAPES ******************************
* Saves the arrays in a disk file.                       *
*        Called from : Main program loop.                *
*******************************************************)
procedure SAVESHAPES;
  begin
    rewrite(SHAPEFILE,'LOGIC.CHARSET');
    SHAPEFILE^:=ARROW; put(SHAPEFILE);
    SHAPEFILE^:=NODEL; put(SHAPEFILE);
    SHAPEFILE^:=NODER; put(SHAPEFILE);
    SHAPEFILE^:=QNODEL; put(SHAPEFILE);
    SHAPEFILE^:=QNODER; put(SHAPEFILE);
    close(SHAPEFILE,lock);
  end;
(************ INIT1 ************************************
* Creates arrays from strings.  The arrow for source/sink*
* nodes.            Called from : Main program loop.     *
*********************************************************)
procedure INIT1;
  begin
    write('.');
    ROW:=20;
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'      X              ');
    MAKESHAPES(ARROW,'       X             ');
    MAKESHAPES(ARROW,'        X            ');
    MAKESHAPES(ARROW,'  X   X  X           ');
    MAKESHAPES(ARROW,'  X   X   X          ');
    MAKESHAPES(ARROW,'XXXXXXXXXXX          ');
    MAKESHAPES(ARROW,'    X     X          ');
    MAKESHAPES(ARROW,'    X    X           ');
    MAKESHAPES(ARROW,'        X            ');
    MAKESHAPES(ARROW,'       X             ');
    MAKESHAPES(ARROW,'      X              ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
    MAKESHAPES(ARROW,'                     ');
  end;
(************  INIT2  ************************************
* Creates arrays from strings.  Left half of nodes.      *
*       Called from : Main program loop.                 *
*********************************************************)
procedure INIT2;
  begin
    write('.');
    ROW:=20;
    MAKESHAPES(NODEL,'         XXXXXXXXXXXX');
    MAKESHAPES(NODEL,'      XXX         X  ');
    MAKESHAPES(NODEL,'    XXX           X  ');
    MAKESHAPES(NODEL,'   XX             X  ');
    MAKESHAPES(NODEL,'  XX              X  ');
    MAKESHAPES(NODEL,' XX               X  ');
    MAKESHAPES(NODEL,' X                X  ');
    MAKESHAPES(NODEL,'XX                X  ');
    MAKESHAPES(NODEL,'X                 X  ');
    MAKESHAPES(NODEL,'X                 X  ');
    MAKESHAPES(NODEL,'XXXXXXXXXXXXXXXXXXXXX');
    MAKESHAPES(NODEL,'X                 X  ');
    MAKESHAPES(NODEL,'X                 X  ');
    MAKESHAPES(NODEL,'XX                X  ');
    MAKESHAPES(NODEL,' X                X  ');
    MAKESHAPES(NODEL,' XX               X  ');
    MAKESHAPES(NODEL,'  XX              X  ');
    MAKESHAPES(NODEL,'   XX            X  ');
    MAKESHAPES(NODEL,'    XXX          X  ');
    MAKESHAPES(NODEL,'      XXX        X  ');
    MAKESHAPES(NODEL,'         XXXXXXXXXXX');
  end;
(************  INIT3  **************************************
* Creates arrays from strings.  Right half of nodes.       *
*        Called from : Main program loop.                  *
***********************************************************)
procedure INIT3;
  begin
    write('.');
    ROW:=20;
    MAKESHAPES(NODER,'XXXXXXXXXXX         ');
    MAKESHAPES(NODER,'      X    XXX      ');
    MAKESHAPES(NODER,'      X      XXX    ');
    MAKESHAPES(NODER,'      X        XX   ');
    MAKESHAPES(NODER,'      X         XX  ');
    MAKESHAPES(NODER,'      X          XX ');
    MAKESHAPES(NODER,'      X           X ');
    MAKESHAPES(NODER,'      X           XX');
    MAKESHAPES(NODER,'      X            X');
    MAKESHAPES(NODER,'      X            X');
    MAKESHAPES(NODER,'XXXXXXX            X');
    MAKESHAPES(NODER,'      X            X');
    MAKESHAPES(NODER,'      X            X');
    MAKESHAPES(NODER,'      X           XX');
    MAKESHAPES(NODER,'      X           X ');
    MAKESHAPES(NODER,'      X          XX ');
    MAKESHAPES(NODER,'      X         XX  ');
    MAKESHAPES(NODER,'      X        XX   ');
    MAKESHAPES(NODER,'      X      XXX    ');
    MAKESHAPES(NODER,'      X    XXX      ');
    MAKESHAPES(NODER,'XXXXXXXXXXX         ');
  end;
(************  INIT4  **************************************
* Creates arrays from strings.  Left half of queue node. *
*        Called from : Main program loop.                  *
***********************************************************)
procedure INIT4;
  begin
    write('.');
    ROW:=20;
    MAKESHAPES(QNODEL,'         XXXXXXXXXXX');
    MAKESHAPES(QNODEL,'      XXX        X  ');
    MAKESHAPES(QNODEL,'    XXX          X  ');
    MAKESHAPES(QNODEL,'   XX            X  ');
    MAKESHAPES(QNODEL,'  XX             X  ');
    MAKESHAPES(QNODEL,' XX              X  ');
    MAKESHAPES(QNODEL,' X               X  ');
    MAKESHAPES(QNODEL,'XX               X  ');
    MAKESHAPES(QNODEL,'X                X  ');
    MAKESHAPES(QNODEL,'X                 X  ');
    MAKESHAPES(QNODEL,'XXXXXXXXXXXXXXXXXXX  ');
    MAKESHAPES(QNODEL,'X                 X  ');
    MAKESHAPES(QNODEL,'X                 X  ');
    MAKESHAPES(QNODEL,'XX                X  ');
    MAKESHAPES(QNODEL,' X                X  ');
    MAKESHAPES(QNODEL,' XX               X  ');
    MAKESHAPES(QNODEL,'  XX              X  ');
    MAKESHAPES(QNODEL,'   XX             X  ');
    MAKESHAPES(QNODEL,'    XXX           X  ');
    MAKESHAPES(QNODEL,'      XXX         X  ');
    MAKESHAPES(QNODEL,'         XXXXXXXXXXXX');
  end;
(************ INIT5 ************************************
* Creates arrays from strings.  Right half of queue node.*
*       Called from : Main program loop.                 *
*********************************************************)
procedure INIT5;
  begin
    write('.');
    ROW:=20;
    MAKESHAPES(QNODER,'XXXXXXXXXXXX         ');
    MAKESHAPES(QNODER,'       X    XXX      ');
    MAKESHAPES(QNODER,'       X      XXX    ');
    MAKESHAPES(QNODER,'       X        XX   ');
    MAKESHAPES(QNODER,'       X         XX  ');
    MAKESHAPES(QNODER,'       X          XX ');
    MAKESHAPES(QNODER,'       X           X ');
    MAKESHAPES(QNODER,'       X           XX');
    MAKESHAPES(QNODER,'       X            X');
    MAKESHAPES(QNODER,'       X            X');
    MAKESHAPES(QNODER,'       X            X');
    MAKESHAPES(QNODER,'       X            X');
    MAKESHAPES(QNODER,'       X            X');
    MAKESHAPES(QNODER,'       X           XX');
    MAKESHAPES(QNODER,'       X       X   X ');
    MAKESHAPES(QNODER,'       X       XX XX ');
    MAKESHAPES(QNODER,'       X        XXX  ');
    MAKESHAPES(QNODER,'       X        XXX  ');
    MAKESHAPES(QNODER,'       X      XXX XX ');
    MAKESHAPES(QNODER,'       X    XXX    XX');
    MAKESHAPES(QNODER,'XXXXXXXXXXXX        X');
  end;
(************ MAIN PROGRAM ****************************)
  begin
    write('initializing array');
    INIT1; INIT2; INIT3; INIT4;INIT5;
    writeln; writeln('Writing QGERT symbols to disk');
    SAVESHAPES;
  end.
```

<u>Appendix D</u>
<u>Program QGERTNET</u>

```
(*$S+*)
unit GLOBAL;  (* saved as UGLOBAL.TEXT *)
intrinsic code 26 data 27;

interface

(**************************************************
* Link lists to store inforamtion for QGERT       *
* source code generation.                         *
**************************************************)
type
   SHAPE = packed array [0..20,0..20] of boolean;

(******  Source and Regular nodes  ***************)
   LINK = ^SOUREG;
   SOUREG = record
      NEXT          : LINK;
      DEVICE        : integer;
      XLOCATION     : integer;
      YLOCATION     : integer;
      TIPE          : string[3];
      NODENUM       : string[2];
      INITIAL       : string[1];
      SUBSEQUENT    : string[1];
      BRANCHING     : string[1];
      MARK          : string[1];
      CHOICE        : string[1];
      COMMENT       : string[25];
   end;

(******  Stat and Sink nodes  ********************)
   LINK2 = ^STASIN;
   STASIN = record
      NEXT          : LINK2;
      DEVICE        : integer;
      XLOCATION     : integer;
      YLOCATION     : integer;
      TIPE          : string[3];
      NODENUM       : string[2];
      INITIAL       : string[1];
      SUBSEQUENT    : string[1];
      BRANCHING     : string[1];
      STAT          : string[1];
      UPPER         : string[4];
      WIDTH         : string[4];
      CHOICE        : string[1];
```

```
        COMMENT       : string[25];
      end;

    (******  Activities  ****************************)
      LINK3 = ^ACT;
      ACT = record
         NEXT         : LINK3;
         DEVICE       : integer;
         SX           : integer;
         SY           : integer;
         EX           : integer;
         EY           : integer;
         TIPE         : string[3];
         START        : string[2];
         IND          : string[2];
         DISTR        : string[2];
         PARAM        : string[4];
         ACTNUM       : string[2];
         SERVERS      : string[2];
         COMMENT      : string[25];
      end;

    (******  Parameter Sets  ************************)
      LINK4 = ^PAR;
      PAR = record
         NEXT         : LINK4;
         TIPE         : string[3];
         PARAM        : string[4];
         PAR1         : string[4];
         PAR2         : string[4];
         PAR3         : string[4];
         PAR4         : string[4];
         COMMENT      : string[25];
      end;

    (******  Queue nodes  ***************************)
      LINK5 = ^QUE;
      QUE = record
         NEXT         : LINK5;
         DEVICE       : integer;
         XLOCATION    : integer;
         YLOCATION    : integer;
         TIPE         : string[3];
         NODENUM      : string[2];
         INITIAL      : string[1];
         CAPACITY     : string[1];
         BRANCHING    : string[1];
         RANKING      : string[1];
         BALKERS      : string[2];
         UPPER        : string[4];
         WIDTH        : string[4];
         COMMENT      : string[25];
    end;
    SOUREGFIL  = file of SOUREG;
    STASINFIL  = file of STASIN;
    ACTFIL     = file of ACT;
    PARFIL     = file of PAR;
    QUEFIL     = file of QUE;

(***********************************************
*           integer variables                  *
***********************************************)
var
        PEN,                          (* pen switch (up or down)      *)
        X,Y,                          (* pen position on pad          *)
        D,                            (* device being plotted         *)
        LASTX,LASTY,           (* last Device,X,&Y            *)
        DMODE                         (* mode used for plotting       *)
                                      : integer;

(***********************************************
* QGERT shape names, used to draw QGERT symbols   *
***********************************************)
       ARROW,NODEL,NODER,QNODEL,QNODER : SHAPE;
(***********************************************
* disk file of QGERT symbols                   *
***********************************************)
       SHAPEFILE    : file of SHAPE;
(***********************************************
*         booleans                             *
***********************************************)
        UPDATE,              (* checks for SAVE in EDITor          *)
        LOCKX,LOCKY,         (* contains the LOCKed x or y coord.  *)
        SAFETY,              (* checks for SAVE on exit            *)
        VALIDXY,             (* true if X & Y are on screen        *)
        HELLFREEZESOVER,     (* never true - for infinite repeats  *)
        INVERSE,
        FINDER
                             : boolean;
(***********************************************
*         strings and things                   *
***********************************************)
        FILENAME, DUMMY,          (* for LOAD and SAVE names      *)
        IDENT,                    (* for names of plotted devices *)
        ACTSTRING,                (* builds Activity data structure*)
        STRSELEC,ELEMENT,         (* used in EDITor searches      *)
        ST                        (* used in WRITEONSCREEN        *)
                             : string;
        CH,SELEC,ANS              (* for inputs and control       *)
                             : char;
        NEXT1,BASE1          :LINK;  (* SOU/REG node pointers  *)
        BASE2,NEXT2          :LINK2; (* STA/SIN node pointers  *)
        BASE3,NEXT3          :LINK3; (* ACT pointers           *)
        BASE4,NEXT4          :LINK4; (* PAR pointers           *)
```

```
          BASE5,NEXT5         :LINK5; (* QUE node pointers      *)
          SRF                 :SOUREGFIL;  (* files of the records *)
          SSF                 :STASINFIL;
          AF                  :ACTFIL;
          PF                  :PARFIL;
          QF                  :QUEFIL;

procedure STUPID;

implementation

   procedure STUPID;  begin end;

begin end.
```

```
(*******************************************************
* This unit is linked to QGERTNET.  It contains the     *
* graphics screen text writer and the QGERT shapes      *
* loader.                                               *
*   Both procedures written by : Dan Sokol              *
*   WRITEONSCREEN modified by  : Anderson & Commeford   *
*******************************************************)

(*$S+*)
unit RITE;   (* saved as UWRITE.TEXT*)
intrinsic code 17;

interface

uses TURTLEGR,GLOBAL;

procedure WRITEONSCREEN;
procedure GETSHAPES;

implementation

(******************  WRITEONSCREEN   ******************
*   Overlays strings on HIRES screen 1.                 *
*   Text can be horiz or vertical.                      *
*   Called by : Units NODEINFO,ACTPAR,UINN              *
******************************************************)

procedure WRITEONSCREEN;
        var I  : integer;
            C1 : char;
        begin
          DMODE := 6; if INVERSE then DMODE := 5;
          chartype(DMODE);
          pencolor(none);
          while VALIDXY do begin
            case CH of
            'h','H': begin moveto(X,Y); wstring(ST);  end;
            'v','V': begin for I:=1 to length(ST) do
                     begin moveto(X,Y); C1:=ST[I]; wchar(C1); Y:=Y-9;  end;
                     end;
            end;
            VALIDXY:=false;
            end; (* of while loop *)
          end;
```

```
(******************** GETSHAPES *************************
* Loads the shapes from the file 'LOGIC.CHARSET'          *
*   Called from: Main program loop.                       *
*******************************************************)
procedure GETSHAPES;
        begin
           reset(SHAPEFILE,'LOGIC.CHARSET');
           ARROW:=SHAPEFILE^; get(SHAPEFILE);
           NODEL:=SHAPEFILE^; get(SHAPEFILE);
           NODER:=SHAPEFILE^; get(SHAPEFILE);
           QNODEL:=SHAPEFILE^; get(SHAPEFILE);
           QNODER:=SHAPEFILE^;
           close(SHAPEFILE);
        end;

begin    end.
```

```
(*******************************************************
* This unit is linked to QGERTNET.  It contains the    *
* the procedures which build the data structure by ask- *
* ing the user questions.  It has procedures for the    *
* source/regular nodes, stat/sink nodes, and queue nodes*
*       Written by : Anderson & Commeford               *
*******************************************************)

(*$S+*)
unit NODEINFO;    (* saved as UNODEINFO.TEXT *)
intrinsic code 25;

interface

uses TURTLEGR,GLOBAL,RITE;

procedure SOUREGINFO;
procedure STASININFO;
procedure QUEINFO;

implementation

(***************  SOUREGINFO  **************************
* Builds the data structure for a source or regular node*
* Also writes text to the graphics screen.             *
*      Called by: MYPLOT                               *
*******************************************************)

procedure SOUREGINFO;
   begin
      new(NEXT1);
      Y:=Y+10;
      NEXT1^.DEVICE:=D; NEXT1^.XLOCATION:=X;
      NEXT1^.YLOCATION:=Y;
      if D = 1 then
         begin NEXT1^.TIPE:='SOU';
            X := X+40; Y := Y+1
         end else begin
            NEXT1^.TIPE:='REG';
            X:=X+30; Y:=Y+1
         end;
      write(chr(12)); CH:='V'; writeln;
      write('Enter node number (1 to 99) ---> ');
      readln(NEXT1^.NODENUM);
      ST:=NEXT1^.NODENUM; WRITEONSCREEN; Y:=NEXT1^.YLOCATION+1;
      X:=X-20; CH:='H'; VALIDXY:=true;
      write('Initial number to release (0 to 9) ---> ');
      readln(NEXT1^.INITIAL);
      ST:=NEXT1^.INITIAL; WRITEONSCREEN;
      Y:=Y-10; VALIDXY:=true;
      write('Subsequent number to release, ( <CR> ');
        write('for infinity or 1 to 9) ---> ');
        readln(NEXT1^.SUBSEQUENT); ST:=NEXT1^.SUBSEQUENT;
        if length(ST) = 0 then ST := '-';
        WRITEONSCREEN; NEXT1^.BRANCHING:='D';
        X:=X+10; VALIDXY:=true;
        write('Enter M to mark, otherwise just press <CR> ---> ');
        readln(NEXT1^.MARK); ST:=NEXT1^.MARK; WRITEONSCREEN;
        Y:=Y+10; VALIDXY:=true;
        write('Enter choice criterion (F,L,S,B,M) --->  ');
        readln(NEXT1^.CHOICE); ST:=NEXT1^.CHOICE; WRITEONSCREEN;
        write('Enter a node comment. <CR> for none ---> ');
        readln(NEXT1^.COMMENT);
        NEXT1^.NEXT := BASE1; BASE1 := NEXT1;
     end;

(**************  STASININFO  *************************
* Builds the data structure for a stat or sink node.    *
* Also writes text to the graphics screen.               *
*      Called by: MYPLOT                                 *
*****************************************************)

procedure STASININFO;
     begin
        new(NEXT2);
        Y:=Y+10;
        NEXT2^.DEVICE:=D; NEXT2^.XLOCATION:=X;
        NEXT2^.YLOCATION:=Y;
        if D = 3
           then NEXT2^.TIPE:='STA'
           else
           NEXT2^.TIPE:='SIN';
        X:=X+30; Y:=Y+1;
        write(chr(12)); CH:='V';
        write('Enter node number (1 to 99) ---> ');
        readln(NEXT2^.NODENUM);
        ST:=NEXT2^.NODENUM; WRITEONSCREEN; Y:=NEXT2^.YLOCATION+1;
        X:=X-20; CH:='H';  VALIDXY:=true;
        write('Initial number to release (0 to 9) ---> ');
        readln(NEXT2^.INITIAL); ST:=NEXT2^.INITIAL;
        WRITEONSCREEN;
        Y:=Y-10;  VALIDXY:=true;
        write('Subsequent number to release, ( <CR> ');
        write('for infinity or 1 to 9) ---> ');
        readln(NEXT2^.SUBSEQUENT); ST:=NEXT2^.SUBSEQUENT;
        if length(ST) = 0 then ST := '-';
        WRITEONSCREEN;
        NEXT2^.BRANCHING:='D';
        X:=X+10; VALIDXY:=true;
        write('Enter statistics desired (F,A,B,I,or D) ---> ');
        readln(NEXT2^.STAT); ST:=NEXT2^.STAT; WRITEONSCREEN;
        write('Enter upper limit of first cell ');
        write('(N if histogram not wanted) --->  ');
        readln(NEXT2^.UPPER);
         if (NEXT2^.UPPER='N')
           then NEXT2^.WIDTH:='N'
           else begin
             write('Enter width of histogram cell --->  ');
             readln(NEXT2^.WIDTH);
           end;  (* of else *)
        Y:=Y+10; VALIDXY:=true;
        write('Enter choice criterion (F,L,S,B,M) --->  ');
        readln(NEXT2^.CHOICE); ST:=NEXT2^.CHOICE; WRITEONSCREEN;
        write('Enter a node comment. <CR> for none ---> ');
        readln(NEXT2^.COMMENT);
        NEXT2^.NEXT := BASE2; BASE2 := NEXT2;
      end;

(**************  QUEINFO  *****************************
* Builds the data structure for a queue node.         *
* Also writes text to the graphics screen.            *
*      Called by: MYPLOT                              *
******************************************************)

procedure QUEINFO;
   begin
      new(NEXT5);
      Y:=Y+10;
      NEXT5^.DEVICE:=D; NEXT5^.XLOCATION:=X;
      NEXT5^.YLOCATION:=Y;
      NEXT5^.TIPE:='QUE';
      X:=X+30; Y:=Y+1;
      write(chr(12)); CH:='V';
      write('Enter node number (1 to 99) ---> ');
      readln(NEXT5^.NODENUM);
      ST:=NEXT5^.NODENUM; WRITEONSCREEN; Y:=NEXT5^.YLOCATION+1;
      X:=X-20; CH:='H';  VALIDXY:=true;
      write('Initial number in queue (0 to 9) ---> ');
      readln(NEXT5^.INITIAL); ST:=NEXT5^.INITIAL;
      WRITEONSCREEN;
      Y:=Y-10;  VALIDXY:=true;
      write('Capacity of queue node, ( <CR> ');
      write('for infinity or 1 to 9) ---> ');
      readln(NEXT5^.CAPACITY); ST:=NEXT5^.CAPACITY;
       if length(ST) = 0 then ST := '-';
      WRITEONSCREEN;
      NEXT5^.BRANCHING:='D';
      X:=X+10; Y:=Y+5; VALIDXY:=true;
      write('Enter type ranking desired (F,L,S,or B) ---> ');
      readln(NEXT5^.RANKING); ST:=NEXT5^.RANKING; WRITEONSCREEN;
      write('Enter balking node number ( <CR> to destroy ');
      write('balkers --->  '); readln(NEXT5^.BALKERS);
      write('Enter upper limit of first cell ');
      write('(N if histogram not wanted) --->  ');
      readln(NEXT5^.UPPER);
```

```
        if (NEXT5^.UPPER='N')
          then NEXT5^.WIDTH:='N'
          else begin
            write('Enter width of histogram cell --->  ');
            readln(NEXT5^.WIDTH);
          end;  (* of else *)
        write('Enter a node comment. <CR> for none ---> ');
        readln(NEXT5^.COMMENT);
        NEXT5^.NEXT := BASE5; BASE5 := NEXT5;
      end;

begin    end.
```

```
(*****************************************************
* This unit is linked to QGERTNET.  It contains the     *
* procedures to build the data structure for activities *
* and parameter sets.                                   *
*       Written by : Anderson & Commeford               *
*****************************************************)
(*$S+*)
unit ACTPAR; (* saved as UACTPAR.TEXT *)
intrinsic code 29;

interface

uses TURTLEGR,GLOBAL,RITE;
procedure ACTINFO;
procedure PARINFO;

implementation

(***************  PARINFO  **************************
* Builds the data structure for a parameter set.    *
*     Called by : ACTINFO                           *
*************************************************)
procedure PARINFO;
begin
   writeln;writeln('       Building Parameter Card');writeln;
   write('Enter parameter set number ---> ');
   readln(NEXT3^.PARAM);
   ST:=NEXT3^.PARAM; ACTSTRING:=concat(ACTSTRING,ST,')');
   NEXT4:=BASE4; FINDER:=false;
   while (not FINDER) and (NEXT4 <> nil) do begin
      if (NEXT4^.PARAM = ST)
         then FINDER:=true
         else NEXT4:=NEXT4^.NEXT;
   end; (* of while *)
   if (not FINDER) then begin
      NEW(NEXT4);
      NEXT4^.TIPE:='PAR';
      NEXT4^.PARAM:=ST;
      write('Enter 1ST parameter ---> ');readln(NEXT4^.PAR1);
      write('Enter 2ND parameter ---> ');readln(NEXT4^.PAR2);
      write('Enter 3RD parameter ---> ');readln(NEXT4^.PAR3);
      write('Enter 4TH parameter ---> ');readln(NEXT4^.PAR4);
      write('Enter your parameter card comment. ');
      write(' <CR> for none ---> ');
      readln(NEXT4^.COMMENT);writeln;
      NEXT4^.NEXT:=BASE4; BASE4:=NEXT4;
   end  (* of FINDER if *)
      else begin writeln;
      writeln('There"s already a parameter set ',ST);
      writeln('Therefore, a parameter card will not be built');
```

```
         writeln; end;
      end;


      (***************  SEARCH  ****************************
      * Finds the start and end node of an activity.         *
      *       Called by : ACTINFO                            *
      ****************************************************)

      procedure SEARCH;
      begin
         NEXT1:=BASE1;
         while (not FINDER) and (NEXT1 <> nil) do begin
            if (NEXT1^.NODENUM = NEXT3^.START) then
               begin FINDER := true;
                  case NEXT1^.DEVICE of
                     1: begin NEXT3^.SX:=NEXT1^.XLOCATION + 53;
                        NEXT3^.SY:=NEXT1^.YLOCATION; end;
                     2: begin NEXT3^.SX:=NEXT1^.XLOCATION + 42;
                        NEXT3^.SY:=NEXT1^.YLOCATION; end;
                  end; (* of case stmt *)
               end
               else NEXT1:=NEXT1^.NEXT;
         end;   (* of while *)
         if (not FINDER) then begin
            NEXT2:=BASE2;
            while (not FINDER) and (NEXT2 <> nil) do begin
               if (NEXT2^.NODENUM = NEXT3^.START) then
                  begin FINDER:=true;
                     case NEXT2^.DEVICE of
                        3: begin NEXT3^.SX:=NEXT2^.XLOCATION + 42;
                           NEXT3^.SY:=NEXT2^.YLOCATION; end;
                        6: begin write(chr(12));
                           write('A sink node can not start a act,');
                           write(' press RETURN');
                           read(CH);exit(ACTINFO);end;
                     end; (* of case stmt *)
                  end
                  else NEXT2:=NEXT2^.NEXT;
            end; (* of while *)
          end;
         if (not FINDER) then begin
            NEXT5:=BASE5;
            while (not FINDER) and (NEXT5 <> nil) do begin
               if (NEXT5^.NODENUM = NEXT3^.START) then
                  begin FINDER:=true;
                        NEXT3^.SX:=NEXT5^.XLOCATION + 42;
                        NEXT3^.SY:=NEXT5^.YLOCATION; end
                  else NEXT5:=NEXT5^.NEXT;
            end; (* of while *)
          end;
          if (not FINDER) then
          begin write(chr(12));
             write('Start node not found, press RETURN');
             read(CH); exit(ACTINFO);
          end;
       pencolor(NONE); moveto(NEXT3^.SX,NEXT3^.SY);
       write('End node number ---> '); readln(NEXT3^.IND);
       FINDER:=false; NEXT1:=BASE1;
       while (not FINDER) and (NEXT1 <> nil) do begin
       if (NEXT1^.NODENUM = NEXT3^.IND) then
          begin FINDER:=true;
             NEXT3^.EX:=NEXT1^.XLOCATION;
             NEXT3^.EY:=NEXT1^.YLOCATION;
          end
          else NEXT1:=NEXT1^.NEXT;
        end; (* of while *)
        if (not FINDER) then
           begin NEXT2:=BASE2;
              while (not FINDER) and (NEXT2 <> nil) do begin
                 if (NEXT2^.NODENUM = NEXT3^.IND) then
                    begin FINDER:=true;
                       case NEXT2^.DEVICE of
                          3,6: begin NEXT3^.EX:=NEXT2^.XLOCATION;
                                NEXT3^.EY:=NEXT2^.YLOCATION;end;
                       end;   (* of case stmt *)
                    end
                    else NEXT2:=NEXT2^.NEXT;
              end;     (* end of while *)
           end;
      if (not FINDER) then begin
         NEXT5:=BASE5;
         while (not FINDER) and (NEXT5 <> nil) do begin
            if (NEXT5^.NODENUM = NEXT3^.IND) then
               begin FINDER:=true;
                     NEXT3^.EX:=NEXT5^.XLOCATION;
                     NEXT3^.EY:=NEXT5^.YLOCATION; end
               else NEXT5:=NEXT5^.NEXT;
         end; (* of while *)
       end;
         if (not FINDER) then
            begin write(chr(12));
               write('End node not found, press RETURN');
               readln(CH);exit(ACTINFO);
            end;
         pencolor(WHITE);
         if (NEXT3^.START <> NEXT3^.IND) then FINDER:=false;
   end;
```

```
(***************  ACTINFO  **************************
* Builds the data structure for an activity.        *
* Also writes text to the graphics screen.          *
*     Called by : MYPLOT                            *
****************************************************)
procedure ACTINFO;
var      TEMPX   :   integer;
begin
   new(NEXT3); NEXT3^.DEVICE:=D; NEXT3^.TIPE:='ACT';
   write(chr(12)); write('Device type >>  ACTIVITY');
   writeln; writeln; writeln;
   write('Start node number ---> '); readln(NEXT3^.START);
   FINDER:=false;   SEARCH;
   if (not FINDER) then
   begin moveto(NEXT3^.EX,NEXT3^.EY);
      X:=(NEXT3^.SX + NEXT3^.EX) div 2;
      Y:=(NEXT3^.SY + NEXT3^.EY) div 2;
   end else
      begin  moveto(NEXT3^.SX,NEXT3^.SY+25);
        moveto(NEXT3^.SX-42,NEXT3^.SY+25);
        moveto(NEXT3^.SX-42,NEXT3^.SY);
        X:=NEXT3^.SX-21; Y:=NEXT3^.SY+25;
      end;
    INVERSE:=true;
    write('Enter distribution type ---> ');
    readln(NEXT3^.DISTR);
    ACTSTRING:=concat('(',NEXT3^.DISTR,',');
    if (NEXT3^.DISTR='CO') then
       begin write('Enter constant value ---> ');
          readln(NEXT3^.PARAM);
          ACTSTRING:=concat(ACTSTRING,NEXT3^.PARAM,')');
       end else PARINFO;
    ST:=ACTSTRING;TEMPX:=X;
    X:=X - ((length(ST) div 2)*7);
    CH:='H'; WRITEONSCREEN;
    write('Enter activity number (1 to 99) ---> ');
    readln(NEXT3^.ACTNUM);
    ACTSTRING:=concat('[',NEXT3^.ACTNUM,'] ');
    write('Enter number of parallel servers (1 to 99) ---> ');
    readln(NEXT3^.SERVERS);
    ACTSTRING:=concat(ACTSTRING,'(',NEXT3^.SERVERS,')');
    ST:=ACTSTRING;
    Y:=Y - 9;X:=TEMPX - ((length(ST) div 2)*7);
    CH:='H'; VALIDXY:=true; WRITEONSCREEN; INVERSE:=false;
    write('Enter activity card comment. (CR) for none ---> ');
    readln(NEXT3^.COMMENT);
    NEXT3^.NEXT:=BASE3;BASE3:=NEXT3;
  end;
begin    end.
```

```
(*******************************************************
* This unit is linked to QGERTNET.  It loads a network's*
* data structure from disk and simultaneously draws the *
* the network on the graphics screen.                   *
*       Written by : Anderson & Commeford               *
*******************************************************)

(*$S+*)
unit INN; (* saved as UINN *)

interface

uses TURTLEGR,GLOBAL,RITE;

procedure INDATA(var SRFF: SOUREGFIL;
                 var SSFF: STASINFIL;
                 var AFF : ACTFIL;
                 var PFF : PARFIL;
                 var QFF : QUEFIL);

procedure LOADSCREEN;

implementation

(**************  GRAFNODES  ***********************
* As each node record is read in from disk, this draws*
* the node on the graphics screen.                   *
*       Called by : INDATA                           *
****************************************************)

procedure GRAFNODES;
   begin pencolor(white); DMODE := 10;
      case D of
         1,2: begin D:= NEXT1^.DEVICE;
                 X := NEXT1^.XLOCATION;
                 Y := NEXT1^.YLOCATION - 10;
              end;
         3,6: begin D:= NEXT2^.DEVICE;
                 X := NEXT2^.XLOCATION;
                 Y := NEXT2^.YLOCATION - 10;
              end;
         5  : begin D:=NEXT5^.DEVICE;
                 X:=NEXT5^.XLOCATION;
                 Y:=NEXT5^.YLOCATION - 10;
              end;
      end;    (* of case stmt *)
      case D of
         1:   begin drawblock(ARROW,4,0,0,21,21,X,Y,DMODE);
                    drawblock(NODEL,4,0,0,21,21,X+11,Y,DMODE);
                    drawblock(NODER,4,0,0,21,21,X+32,Y,DMODE);
              end;
```

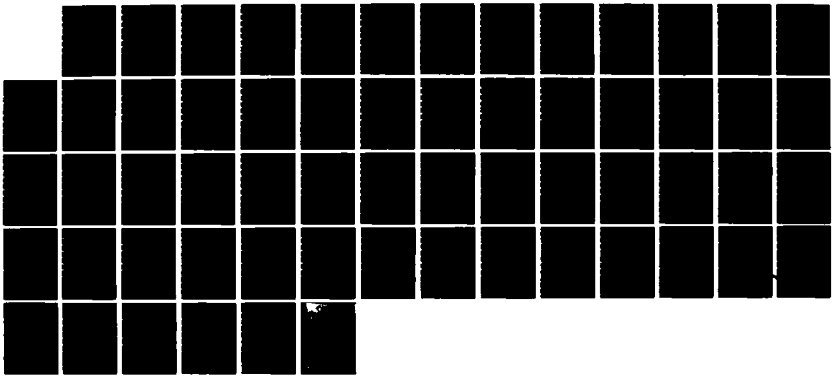
```
    2,3: begin drawblock(NODEL,4,0,0,21,21,X,Y,DMODE);
               drawblock(NODER,4,0,0,21,21,X+21,Y,DMODE);
         end;
    5:   begin drawblock(QNODEL,4,0,0,21,21,X,Y,DMODE);
               drawblock(QNODER,4,0,0,21,21,X+21,Y,DMODE);
         end;
    6:   begin drawblock(NODEL,4,0,0,21,21,X,Y,DMODE);
               drawblock(NODER,4,0,0,21,21,X+21,Y,DMODE);
               drawblock(ARROW,4,0,0,21,21,X+42,Y,DMODE);
         end;
    end;  (* of case stmt *)

X := X + 30; Y := Y + 11; CH := 'V';
if D = 1 then X := X + 10;
case D of
    1,2: ST := NEXT1^.NODENUM;
    5  : ST:=NEXT5^.NODENUM;
    3,6: ST := NEXT2^.NODENUM;
end;  (* of case stmt *)
VALIDXY := true; WRITEONSCREEN;
X := X - 20; CH := 'H';
case D of
    1,2: begin Y := NEXT1^.YLOCATION + 1;
           ST := NEXT1^.INITIAL;
         end;
    5  : begin Y:=NEXT5^.YLOCATION + 1;
           ST:=NEXT5^.INITIAL;
         end;
    3,6: begin Y := NEXT2^.YLOCATION + 1;
           ST := NEXT2^.INITIAL;
         end;
end;  (* of case stmt *)
VALIDXY := true; WRITEONSCREEN; Y := Y -10;
case D of
    1,2: ST := NEXT1^.SUBSEQUENT;
    5  : ST:=NEXT5^.CAPACITY;
    3,6: ST := NEXT2^.SUBSEQUENT;
end;  (* of case stmt *)
if (length(ST)=0) then ST:='-';
VALIDXY := true; WRITEONSCREEN; X := X + 10;
case D of
    1,2: ST := NEXT1^.MARK;
    5  : begin Y:=Y + 5; ST:=NEXT5^.RANKING; end;
    3,6: ST := NEXT2^.STAT;
end;  (* of case stmt *)
VALIDXY := true; WRITEONSCREEN;
case D of
    1,2: begin  ST:=NEXT1^.CHOICE;
           Y:=Y+10; VALIDXY:=true;
           WRITEONSCREEN;
         end;
    3,6: begin  ST:=NEXT2^.CHOICE;
```

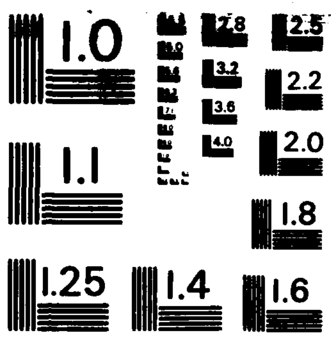
```
                  Y:=Y+10; VALIDXY:=true;
                  WRITEONSCREEN;
                end;
          end;  (* of case stmt *)
      end;

(**************  GRAFACTIVITY  *********************
* As an activity record is read in, this draws the    *
* activity on the graphics screen.                    *
*       Called by : INDATA                            *
*****************************************************)

 procedure GRAFACTIVITY;
   var   TEMPX    : integer;

   begin   DMODE:=10; pencolor(none);
      moveto(NEXT3^.SX,NEXT3^.SY);
      pencolor(white);
      if (NEXT3^.START <> NEXT3^.IND) then
         begin   moveto(NEXT3^.EX,NEXT3^.EY);
          X:=(NEXT3^.SX+NEXT3^.EX) div 2;
          Y:=(NEXT3^.SY+NEXT3^.EY) div 2;
          end else
          begin  moveto(NEXT3^.SX,NEXT3^.SY + 25);
                 moveto(NEXT3^.SX-42,NEXT3^.SY+25);
                 moveto(NEXT3^.SX-42,NEXT3^.SY);
                 X:=NEXT3^.SX-21;
                 Y:=NEXT3^.SY+25;
          end;
      ST:=concat('(',NEXT3^.DISTR,',',NEXT3^.PARAM,')');
      TEMPX:=X;
      X:=X-((length(ST) div 2)*7);
      INVERSE:=true; VALIDXY:=true; CH:='H';
      WRITEONSCREEN;
      ST:=concat('[',NEXT3^.ACTNUM,'] (',NEXT3^.SERVERS,')');
      Y:=Y-9; X:=TEMPX-((length(ST) DIV 2)*7);
      VALIDXY:=true; WRITEONSCREEN; INVERSE:=false;
   end;

(**************  INDATA  ********************
* Loads a file into memory as link lists.      *
*       Called by : LOADSCREEN                 *
**********************************************)

procedure INDATA;
var   DUMMY       : string;
      AGAIN       : boolean;

begin write(chr(12));
   if (not UPDATE) then begin
     write('Load what file ---> '); readln(FILENAME);
     if length(FILENAME) =  0 then exit(INDATA);
```

```
        if length(FILENAME) ) 10 then
           begin writeln; writeln('Filename too long!!',chr(7));
              exit(INDATA);
           end;
     end;  (* of then *)
     DUMMY := concat(FILENAME,'.SOUREG'); D := 1;
(*$I-*)
     reset(SRFF,DUMMY);
     if (IORESULT <> 0)
        then begin writeln;
           writeln('File called ',DUMMY,' not found');
           writeln('<CR> to continue');read(CH);
(*$I+*)
           exit(INDATA);end;
     writeln;writeln('Reading ',DUMMY,' from disk');
     BASE1:=nil;
     while not eof(SRFF) do begin
        new(NEXT1);
        NEXT1^ := SRFF^;
        GRAFNODES;
        NEXT1^.NEXT:=BASE1; BASE1:=NEXT1;
        get(SRFF);
     end;
     close(SRFF);

     DUMMY := concat(FILENAME,'.STASIN'); D := 3;
     writeln;writeln('Reading ',DUMMY,' from disk');
     reset(SSFF,DUMMY); BASE2:=nil;
     while not eof(SSFF) do begin
        new(NEXT2);
        NEXT2^ := SSFF^;
        GRAFNODES;
        NEXT2^.NEXT:=BASE2; BASE2:=NEXT2;
        get(SSFF);
     end;
     close(SSFF);

     DUMMY := concat(FILENAME,'.QUE'); D := 5;
     writeln;writeln('Reading ',DUMMY,' from disk');
     reset(QFF,DUMMY); BASE5:=nil;
     while not eof(QFF) do begin
        new(NEXT5);
        NEXT5^ := QFF^;
        GRAFNODES;
        NEXT5^.NEXT:=BASE5; BASE5:=NEXT5;
        get(QFF);
     end;
     close(QFF);

     DUMMY := concat(FILENAME,'.ACT');
     writeln;writeln('Reading ',DUMMY,' from disk');
     reset(AFF,DUMMY); BASE3:=nil;
```

```
   while not eof(AFF) do begin
      new(NEXT3);
      NEXT3^ := AFF^;
      GRAFACTIVITY;
      NEXT3^.NEXT:=BASE3; BASE3:=NEXT3;
      get(AFF);
   end;
   close(AFF);

   DUMMY := concat(FILENAME,'.PAR');
   writeln;writeln('Reading ',DUMMY,' from disk');
   reset(PFF,DUMMY); BASE4:=nil;
   while not eof(PFF) do begin
      new(NEXT4);
      NEXT4^ := PFF^;
      NEXT4^.NEXT := BASE4; BASE4 := NEXT4;
      get(PFF);
   end;
   close(PFF);

end;

(***************  LOADSCREEN  ******************
*       Called by : MENU                       *
**********************************************)

procedure LOADSCREEN;
   begin INDATA(SRF,SSF,AF,PF,QF); end;



begin end.
```

AD-A141 127 AUTOMATED Q-GERT SOURCE CODE GENERATION USING COMPUTER 2/2
AIDED DESIGN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON
AFB OH SCHOOL OF ENGI.. G M ANDERSON ET AL. DEC 83
UNCLASSIFIED AFIT/GOR/OS/83D-2 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
(*****************************************************
* This unit is linked to QGERTNET.  It saves a network  *
* to disk by writing the link lists to different files  *
* which have the same prefix as supplied by the user.   *
* Unit UINN uses these files to load the data structure *
* into memory and draw the graphics                     *
*       Written by : Anderson & Commeford               *
*****************************************************)

(*$S+*)
unit OUT; (* saved as UOUT.TEXT *)

interface

uses GLOBAL;

procedure OUTDATA(var SRFF: SOUREGFIL;
                  var SSFF: STASINFIL;
                  var AFF : ACTFIL;
                  var PFF : PARFIL;
                  var QFF : QUEFIL);
procedure SAVESCREEN;

implementation

(**************  OUTDATA  ***************************
* Writes the link lists in memory to disk with user's *
* filename.     Called by : SAVESCREEN               *
*****************************************************)

procedure OUTDATA;

begin
   if (not UPDATE) then begin
      write(chr(12)); write('Save with what name ---> ');
      readln(FILENAME);
      if length(FILENAME) = 0 then exit(OUTDATA);
      if length(FILENAME) > 10 then
         begin writeln; writeln('Filename is too long!!',chr(7));
            exit(OUTDATA);
         end;
   end;  (* of then *)
   DUMMY := concat(FILENAME,'.SOUREG');
   writeln;writeln('Writing ',DUMMY,' to disk');
   rewrite(SRFF,DUMMY); NEXT1 := BASE1;
   while (NEXT1 <> nil) do begin
      SRFF^ := NEXT1^; put(SRFF); NEXT1 := NEXT1^.NEXT;
   end;   (* of while *)
   close(SRFF,LOCK);

   DUMMY := concat(FILENAME,'.STASIN');
```

```
      writeln;writeln('Writing ',DUMMY,' to disk');
      rewrite(SSFF,DUMMY); NEXT2 := BASE2;
      while (NEXT2 <> nil) do begin
         SSFF^ := NEXT2^; put(SSFF); NEXT2 := NEXT2^.NEXT;
      end;   (* of while *)
      close(SSFF,lock);

      DUMMY := concat(FILENAME,'.ACT');
      writeln;writeln('Writing ',DUMMY,' to disk');
      rewrite(AFF,DUMMY); NEXT3 := BASE3;
      while (NEXT3 <> nil) do begin
         AFF^ := NEXT3^; put(AFF); NEXT3 := NEXT3^.NEXT;
      end;   (* of while *)
      close(AFF,lock);

      DUMMY := concat(FILENAME,'.PAR');
      writeln;writeln('Writing ',DUMMY,' to disk');
      rewrite(PFF,DUMMY); NEXT4 := BASE4;
      while (NEXT4 <> nil) do begin
         PFF^ := NEXT4^; put(PFF); NEXT4 := NEXT4^.NEXT;
      end;   (* of while *)
      close(PFF,lock);

      DUMMY := concat(FILENAME,'.QUE');
      writeln;writeln('Writing ',DUMMY,' to disk');
      rewrite(QFF,DUMMY); NEXT5 := BASE5;
      while (NEXT5 <> nil) do begin
         QFF^ := NEXT5^; put(QFF); NEXT5 := NEXT5^.NEXT;
      end;   (* of while *)
      close(QFF,lock);

      SAFETY := true;
   end;

   (***************   SAVESCREEN   ********************
   *       Called by : MENU                            *
   **************************************************)

   procedure SAVESCREEN;
   begin  OUTDATA(SRF,SSF,AF,PF,QF); end;


   begin end.
```

```
(*******************************************************
* This unit is linked to QGERTNET.  It allows the user  *
* to change or delete source/regular nodes.             *
*       Written by : Anderson & Commeford               *
*******************************************************)

(*$S+*)
unit EDITSR; (* saved as UEDITSR.TEXT *)

interface

uses GLOBAL;

procedure FINDSOUREG;

implementation

(**************  CHNGSOUREG  ****************************
* Allows changes of certain fields to source/regular nodes *
*       Called by : FINDSOUREG                             *
*********************************************************)

procedure CHNGSOUREG;
begin
   writeln; write('To change a field above, type in the ');
   writeln('first letter ');
   writeln('of the field you want to change.');
   writeln('A blank field means the default value.');
   write('You may type a Q when you are done with ');
   writeln('this node.');
   while (ANS <> 'Q') do begin
      repeat
         gotoxy(30,14); write('                              ');
         gotoxy(20,16); write('                              ');
         gotoxy(0,14); write('Enter your choice ');
         write('(N,I,S,M,H,C,Q) ---> ');
         read(ANS);
      until ((ANS='N')or(ANS='I')or(ANS='S')or(ANS='M')
      or(ANS='H')or(ANS='C')or(ANS='Q'));
      if (ANS='Q') then exit(CHNGSOUREG);
      gotoxy(0,16);
      write('Enter new value ---> ');
      case ANS of
         'N': readln(NEXT1^.NODENUM);
         'I': readln(NEXT1^.INITIAL);
         'S': readln(NEXT1^.SUBSEQUENT);
         'M': readln(NEXT1^.MARK);
         'H': readln(NEXT1^.CHOICE);
         'C': readln(NEXT1^.COMMENT);
      end; (* of case stmt *)
      gotoxy(0,2);write('                                    ');
```

```
        write('                                   ');
        gotoxy(3,2); write(NEXT1^.TIPE);
        gotoxy(9,2); write(NEXT1^.NODENUM);
        gotoxy(16,2); write(NEXT1^.INITIAL);
        gotoxy(26,2); write(NEXT1^.SUBSEQUENT);
        gotoxy(39,2); write(NEXT1^.MARK);
        gotoxy(46,2); write(NEXT1^.CHOICE);
        gotoxy(3,4); write(NEXT1^.COMMENT);
     end;  (* of while *)
  end;

  (**************  GOTSOUREG  **********************
  * After finding selected node, displays information *
  * of that node and asks if deletion is desired.     *
  *       Called by : FINDSOUREG                      *
  ***************************************************)

  procedure GOTSOUREG;
  begin
     write(chr(12));
     writeln('Here is your node');
     write(' TYPE  N(ODE  I(NITIAL  S(UBSEQUENT  M(ARK  ');
     write('C(H)OICE');
     gotoxy(0,3); write('  C(OMMENT');
     if FINDER then
       begin
        gotoxy(3,2); write(NEXT1^.TIPE);
        gotoxy(9,2); write(NEXT1^.NODENUM);
        gotoxy(16,2); write(NEXT1^.INITIAL);
        gotoxy(26,2); write(NEXT1^.SUBSEQUENT);
        gotoxy(39,2); write(NEXT1^.MARK);
        gotoxy(46,2); write(NEXT1^.CHOICE);
        gotoxy(3,4); write(NEXT1^.COMMENT);
       end else
       begin
        gotoxy(3,2); write(NEXT1^.NEXT^.TIPE);
        gotoxy(9,2); write(NEXT1^.NEXT^.NODENUM);
        gotoxy(16,2); write(NEXT1^.NEXT^.INITIAL);
        gotoxy(26,2); write(NEXT1^.NEXT^.SUBSEQUENT);
        gotoxy(39,2); write(NEXT1^.NEXT^.MARK);
        gotoxy(46,2); write(NEXT1^.NEXT^.CHOICE);
        gotoxy(3,4); write(NEXT1^.NEXT^.COMMENT);
       end;
     repeat
       gotoxy(48,7); write('  ');
       gotoxy(0,7);
       write('Do you want to delete this entire node? ');
       write('Y/N ---> ');
       read(ANS);
     until ((ANS='Y') or (ANS='N'));
  end;
```

```
(*************** FINDSOUREG ************************
* Finds the selected node and then either deletes the *
* node or calls CHNGSOUREG.                           *
*       Called by : unit EDIT                         *
*****************************************************)

procedure FINDSOUREG;
begin
   if (BASE1 = nil) then
      begin writeln('Can"t find ',STRSELEC,ELEMENT);
         writeln('(CR) to continue'); read(ANS);
         exit(FINDSOUREG);
      end; (* of then *)
   FINDER:=true;
   if (NEXT1^.NODENUM = ELEMENT) then
      begin GOTSOUREG;
         if (ANS = 'Y') then BASE1:=BASE1^.NEXT
           else CHNGSOUREG;
      end
      else begin
         FINDER:=false;
         repeat
            if (NEXT1^.NEXT^.NODENUM <> ELEMENT) then
            NEXT1:=NEXT1^.NEXT;
         until ((NEXT1^.NEXT=nil)
         or (NEXT1^.NEXT^.NODENUM=ELEMENT));
         if (NEXT1^.NEXT=nil) then
            begin writeln('Can"t find ',STRSELEC,ELEMENT);
               writeln('(CR) to continue'); read(ANS);
               exit(FINDSOUREG);
            end; (* of then *)
         GOTSOUREG;
         if (ANS = 'Y') then NEXT1^.NEXT:=NEXT1^.NEXT^.NEXT
            else begin NEXT1:=NEXT1^.NEXT; CHNGSOUREG; end;
      end; (* of else *)
end;

begin   end.
```

```
(*********************************************
* This unit is linked to QGERTNET.  It allows   *
* the user to change/delete a stat/sink node.   *
*       Written by : Anderson & Commeford       *
*********************************************)

(*$S+*)
unit EDITSS; (* saved as UEDITSS.TEXT *)

interface

uses GLOBAL;

procedure FINDSTASIN;

implementation

(*************** CHNGSTASIN *********************
* Allows changes of certain fields to stat/sink    *
* nodes.        Called by : FINDSTASIN             *
***********************************************)

procedure CHNGSTASIN;
begin
   writeln; write('To change a field above, type in the ');
   writeln('first letter ');
   writeln('of the field you want to change.');
   writeln('A blank field means default value.');
   write('You may type a Q when you are done with this');
   writeln(' node.');
   while (ANS <> 'Q') do begin
      repeat
         gotoxy(30,14); write('                                    ');
         gotoxy(20,16); write('                                    ');
         gotoxy(0,14); write('Enter your choice ');
         write('(N,I,S,T,U,W,H,C,Q) ---> ');
         read(ANS);
      until ((ANS='N')or(ANS='I')or(ANS='S')or(ANS='T')
      or(ANS='U')or(ANS='W')or(ANS='H')or(ANS='C')or(ANS='Q'));
      if (ANS='Q') then exit(CHNGSTASIN);
      gotoxy(0,16); write('Enter new value ---> ');
      case ANS of
         'N': readln(NEXT2^.NODENUM);
         'I': readln(NEXT2^.INITIAL);
         'S': readln(NEXT2^.SUBSEQUENT);
         'T': readln(NEXT2^.STAT);
         'U': readln(NEXT2^.UPPER);
         'W': readln(NEXT2^.WIDTH);
         'H': readln(NEXT2^.CHOICE);
         'C': readln(NEXT2^.COMMENT);
      end; (* of case stmt *)
      gotoxy(0,2);write('                                        ');
      write('                                   ');
      gotoxy(3,2); write(NEXT2^.TIPE);
      gotoxy(9,2); write(NEXT2^.NODENUM);
      gotoxy(16,2); write(NEXT2^.INITIAL);
      gotoxy(26,2); write(NEXT2^.SUBSEQUENT);
      gotoxy(39,2); write(NEXT2^.STAT);
      gotoxy(47,2); write(NEXT2^.UPPER);
      gotoxy(55,2); write(NEXT2^.WIDTH);
      gotoxy(63,2); write(NEXT2^.CHOICE);
      gotoxy(3,4); write(NEXT2^.COMMENT);
   end;  (* of while *)
end;

(**************  GOTSTASIN  **************************
* After finding the selected node, displays information *
* of that node and asks if deletion is desired.         *
*       Called by : FINDSTASIN                          *
*********************************************************)

procedure GOTSTASIN;
begin
   write(chr(12));
   writeln('Here is your node');
   write('  TYPE  N(ODE  I(NITIAL  S(UBSEQUENT  S(T)AT  ');
   write('U(PPER  W(IDTH  C(H)OICE');
   gotoxy(0,3); write('  C(OMMENT');
   if FINDER then
     begin
      gotoxy(3,2); write(NEXT2^.TIPE);
      gotoxy(9,2); write(NEXT2^.NODENUM);
      gotoxy(16,2); write(NEXT2^.INITIAL);
      gotoxy(26,2); write(NEXT2^.SUBSEQUENT);
      gotoxy(39,2); write(NEXT2^.STAT);
      gotoxy(47,2); write(NEXT2^.UPPER);
      gotoxy(55,2); write(NEXT2^.WIDTH);
      gotoxy(63,2); write(NEXT2^.CHOICE);
      gotoxy(3,4); write(NEXT2^.COMMENT);
     end else
     begin
      gotoxy(3,2); write(NEXT2^.NEXT^.TIPE);
      gotoxy(9,2); write(NEXT2^.NEXT^.NODENUM);
      gotoxy(16,2); write(NEXT2^.NEXT^.INITIAL);
      gotoxy(26,2); write(NEXT2^.NEXT^.SUBSEQUENT);
      gotoxy(39,2); write(NEXT2^.NEXT^.STAT);
      gotoxy(47,2); write(NEXT2^.NEXT^.UPPER);
      gotoxy(55,2); write(NEXT2^.NEXT^.WIDTH);
      gotoxy(63,2); write(NEXT2^.NEXT^.CHOICE);
      gotoxy(3,4); write(NEXT2^.NEXT^.COMMENT);
     end;
   repeat
     gotoxy(48,7); write('  ');
```

```
      gotoxy(0,7);
      write('Do you want to delete this entire node? ');
      write('Y/N ---> ');
      read(ANS);
    until ((ANS='Y') or (ANS='N'));
end;

(*************** FINDSTASIN ********************
* Finds the selected node and then either deletes *
* the node or calls CHNGSTASIN.                   *
*       Called by : unit EDIT                     *
***************************************************)

procedure FINDSTASIN;
begin
   if (BASE2 = nil) then
      begin writeln('Can't find ',STRSELEC,ELEMENT);
         writeln('<CR> to continue'); read(ANS);
         exit(FINDSTASIN);
      end; (* of then *)
   FINDER:=true;
   if (NEXT2^.NODENUM = ELEMENT) then
      begin GOTSTASIN;
         if (ANS = 'Y') then BASE2:=BASE2^.NEXT
           else CHNGSTASIN;
      end
      else begin
         FINDER:=false;
         repeat
            if (NEXT2^.NEXT^.NODENUM <> ELEMENT) then
            NEXT2:=NEXT2^.NEXT;
         until ((NEXT2^.NEXT=nil) or (NEXT2^.NEXT^.NODENUM=ELEMENT));
         if (NEXT2^.NEXT=nil) then
            begin writeln('Can't find ',STRSELEC,ELEMENT);
               writeln('<CR> to continue'); read(ANS);
               exit(FINDSTASIN);
            end; (* of then *)
         GOTSTASIN;
         if (ANS = 'Y') then NEXT2^.NEXT:=NEXT2^.NEXT^.NEXT
            else begin NEXT2:=NEXT2^.NEXT; CHNGSTASIN; end;
      end; (* of else *)
end;

begin   end.
```

```
(*********************************************
* This unit is linked to QGERTNET.  It allows   *
* change/deletion of queue nodes.               *
*       Written by : Anderson & Conneford       *
*********************************************)

(*$S+*)
unit EDITQUE; (* saved as UEDITQUE.TEXT *)

interface

uses GLOBAL;

procedure FINDQUE;

implementation

(**************  CHNGQUE  ********************
* Allows changes of certain fields to queue     *
* nodes.        Called by : FINDQUE             *
*********************************************)

procedure CHNGQUE;
begin
   writeln; write('To change a field above, type in the ');
   writeln('first letter ');
   writeln('of the field you want to change.  A blank');
   writeln('field means default value.  You may type a');
   writeln('Q when you are done with this node.');
   while (ANS <> 'Q') do begin
      repeat
         gotoxy(30,14); write('                                  ');
         gotoxy(20,16); write('                                  ');
         gotoxy(0,14); write('Enter your choice ');
         write('(N,I,A,R,B,U,W,C,Q) ---> ');
         read(ANS);
      until ((ANS='N')or(ANS='I')or(ANS='A')or(ANS='R')
      or(ANS='B')or(ANS='U')or(ANS='W')or(ANS='C')or(ANS='Q'));
      if (ANS='Q') then exit(CHNGQUE);
      gotoxy(0,16); write('Enter new value ---> ');
      case ANS of
         'N': readln(NEXT5^.NODENUM);
         'I': readln(NEXT5^.INITIAL);
         'A': readln(NEXT5^.CAPACITY);
         'R': readln(NEXT5^.RANKING);
         'B': readln(NEXT5^.BALKERS);
         'U': readln(NEXT5^.UPPER);
         'W': readln(NEXT5^.WIDTH);
         'C': readln(NEXT5^.COMMENT);
      end; (* of case stmt *)
      gotoxy(0,2);write('                                             ');
      write('                                       ');
      gotoxy(3,2); write(NEXT5^.TIPE);
      gotoxy(9,2); write(NEXT5^.NODENUM);
      gotoxy(16,2); write(NEXT5^.INITIAL);
      gotoxy(26,2); write(NEXT5^.CAPACITY);
      gotoxy(38,2); write(NEXT5^.RANKING);
      gotoxy(48,2); write(NEXT5^.BALKERS);
      gotoxy(58,2); write(NEXT5^.UPPER);
      gotoxy(66,2); write(NEXT5^.WIDTH);
      gotoxy(3,4); write(NEXT5^.COMMENT);
   end;  (* of while *)
end;

(**************  GOTQUE  *********************
* After finding selected queue node, displays   *
* information of that node and asks if deletion *
* is desired.          Called by : FINDQUE      *
*********************************************)

procedure GOTQUE;
begin
   write(chr(12));
   writeln('Here is your node');
   write('  TYPE  N(ODE  I(NITIAL  C(A)PACITY  R(ANKING  ');
   write('B(ALKING  U(PPER  W(IDTH');
   gotoxy(0,3); write('  C(OMMENT');
   if FINDER then
     begin
      gotoxy(3,2); write(NEXT5^.TIPE);
      gotoxy(9,2); write(NEXT5^.NODENUM);
      gotoxy(16,2); write(NEXT5^.INITIAL);
      gotoxy(26,2); write(NEXT5^.CAPACITY);
      gotoxy(38,2); write(NEXT5^.RANKING);
      gotoxy(48,2); write(NEXT5^.BALKERS);
      gotoxy(58,2); write(NEXT5^.UPPER);
      gotoxy(66,2); write(NEXT5^.WIDTH);
      gotoxy(3,4); write(NEXT5^.COMMENT);
     end else
     begin
      gotoxy(3,2); write(NEXT5^.NEXT^.TIPE);
      gotoxy(9,2); write(NEXT5^.NEXT^.NODENUM);
      gotoxy(16,2); write(NEXT5^.NEXT^.INITIAL);
      gotoxy(26,2); write(NEXT5^.NEXT^.CAPACITY);
      gotoxy(38,2); write(NEXT5^.NEXT^.RANKING);
      gotoxy(48,2); write(NEXT5^.NEXT^.BALKERS);
      gotoxy(58,2); write(NEXT5^.NEXT^.UPPER);
      gotoxy(66,2); write(NEXT5^.NEXT^.WIDTH);
      gotoxy(3,4); write(NEXT5^.NEXT^.COMMENT);
     end;
   repeat
     gotoxy(48,7); write('  ');
```

```
      gotoxy(0,7);
      write('Do you want to delete this entire node?');
      write(' Y/N ---> ');
      read(ANS);
    until ((ANS='Y') or (ANS='N'));
end;

(**************  FINDQUE  *********************
* Finds the selected queue node and then either *
* deletes the node or calls CHNGQUE.            *
*       Called by : unit EDIT                   *
***********************************************)

procedure FINDQUE;
begin
   if (BASE5 = nil) then
      begin writeln('Can't find ',STRSELEC,ELEMENT);
         writeln('<CR> to continue'); read(ANS);
         exit(FINDQUE);
      end; (* of then *)
   FINDER:=true;
   if (NEXT5^.NODENUM = ELEMENT) then
      begin GOTQUE;
         if (ANS = 'Y') then BASE5:=BASE5^.NEXT
           else CHNGQUE;
      end
      else begin
         FINDER:=false;
         repeat
            if (NEXT5^.NEXT^.NODENUM <> ELEMENT) then
            NEXT5:=NEXT5^.NEXT;
         until ((NEXT5^.NEXT=nil) or (NEXT5^.NEXT^.NODENUM=ELEMENT));
         if (NEXT5^.NEXT=nil) then
            begin writeln('Can't find ',STRSELEC,ELEMENT);
               writeln('<CR> to continue'); read(ANS);
               exit(FINDQUE);
            end; (* of then *)
         GOTQUE;
         if (ANS = 'Y') then NEXT5^.NEXT:=NEXT5^.NEXT^.NEXT
            else begin NEXT5:=NEXT5^.NEXT; CHNGQUE; end;
      end; (* of else *)
end;

begin    end.
```

```
(*********************************************
* This unit is linked to QGERTNET.  It allows   *
* the user to change/delete an activity.        *
*       Written by : Anderson & Commeford       *
*********************************************)

(*$S+*)
unit EDITACT; (* saved as UEDITACT.TEXT *)

interface

uses GLOBAL;

procedure FINDACT;

implementation

(**************  CHNGACT  **********************
* Allows changes of certain fields to activities. *
*       Called by : FINDACT                        *
************************************************)

procedure CHNGACT;
begin
   writeln; write('To change a field above, type in the ');
   writeln('first letter ');
   write('of the field you want to change.  A blank ');
   writeln('field means the');
   writeln('default value.  If you want to change the');
   write('START or END node you must delete the ');
   writeln('activity and then');
   write('add the new one in the main program.  ');
   writeln('You may type a Q when');
   writeln('you are done with this activity.');
   while (ANS <> 'Q') do begin
      repeat
         gotoxy(30,14); write('                              ');
         gotoxy(20,16); write('                              ');
         gotoxy(0,14);
         write('Enter your choice (D,P,A,S,C,Q) ---> ');
         read(ANS);
      until ((ANS='D')or(ANS='P')or(ANS='A')or(ANS='S')
      or(ANS='C')or(ANS='Q'));
      if (ANS='Q') then exit(CHNGACT);
      gotoxy(0,16); write('Enter new value ---> ');
      case ANS of
         'D': readln(NEXT3^.DISTR);
         'P': readln(NEXT3^.PARAM);
         'A': readln(NEXT3^.ACTNUM);
         'S': readln(NEXT3^.SERVERS);
         'C': readln(NEXT3^.COMMENT);
```

```
        end; (* of case stmt *)
        gotoxy(0,2);write('                                            ');
        write('                                     ');
        gotoxy(3,2); write(NEXT3^.TIPE);
        gotoxy(9,2); write(NEXT3^.START);
        gotoxy(16,2); write(NEXT3^.IND);
        gotoxy(21,2); write(NEXT3^.DISTR);
        gotoxy(29,2); write(NEXT3^.PARAM);
        gotoxy(37;2); write(NEXT3^.ACTNUM);
        gotoxy(44,2); write(NEXT3^.SERVERS);
        gotoxy(3,4); write(NEXT3^.COMMENT);
      end;  (* of while *)
    end;

    (**************  GOTACT  ****************************
    * After finding selected node, displays information of  *
    * that node and asks if deletion is desired.            *
    *       Called by : FINDACT                              *
    *******************************************************)

    procedure GOTACT;
    begin
      write(chr(12));
      writeln('Here is your activity');
      writeln(' TYPE  START  END  D(ISTR  P(ARAM  A(CT#  S(ERVERS');
      gotoxy(0,3); write(' C(OMMENT');
      if FINDER then
        begin
          gotoxy(3,2); write(NEXT3^.TIPE);
          gotoxy(9,2); write(NEXT3^.START);
          gotoxy(16,2); write(NEXT3^.IND);
          gotoxy(21,2); write(NEXT3^.DISTR);
          gotoxy(29,2); write(NEXT3^.PARAM);
          gotoxy(37,2); write(NEXT3^.ACTNUM);
          gotoxy(44,2); write(NEXT3^.SERVERS);
          gotoxy(3,4); write(NEXT3^.COMMENT);
      end else
        begin
          gotoxy(3,2); write(NEXT3^.NEXT^.TIPE);
          gotoxy(9,2); write(NEXT3^.NEXT^.START);
          gotoxy(16,2); write(NEXT3^.NEXT^.IND);
          gotoxy(21,2); write(NEXT3^.NEXT^.DISTR);
          gotoxy(29,2); write(NEXT3^.NEXT^.PARAM);
          gotoxy(37,2); write(NEXT3^.NEXT^.ACTNUM);
          gotoxy(44,2); write(NEXT3^.NEXT^.SERVERS);
          gotoxy(3,4); write(NEXT3^.NEXT^.COMMENT);
      end;
      repeat
        gotoxy(52,7); write('  ');
        gotoxy(0,7);
        write('Do you want to delete this entire activity? ');
        write('Y/N ---> ');
      read(ANS);
    until ((ANS='Y') or (ANS='N'));
end;

(**************  FINDACT  ****************************
* Finds the selected node and then either deletes the   *
* node or calls CHNGACT.                                *
*       Called by : unit EDIT                           *
*****************************************************)

procedure FINDACT;
begin
   if (BASE3 = nil) then
      begin writeln('Can't find ',STRSELEC,ELEMENT);
         writeln('(CR) to continue'); read(ANS);
         exit(FINDACT);
      end; (* of then *)
   FINDER:=true;
   if (NEXT3^.ACTNUM = ELEMENT) then
      begin GOTACT;
         if (ANS = 'Y') then BASE3:=BASE3^.NEXT
           else CHNGACT;
      end
      else begin
         FINDER:=false;
         repeat
            if (NEXT3^.NEXT^.ACTNUM <> ELEMENT) then
            NEXT3:=NEXT3^.NEXT;
         until ((NEXT3^.NEXT=nil)
         or (NEXT3^.NEXT^.ACTNUM=ELEMENT));
         if (NEXT3^.NEXT=nil) then
            begin writeln('Can't find ',STRSELEC,ELEMENT);
               writeln('(CR) to continue'); read(ANS);
               exit(FINDACT);
            end; (* of then *)
         GOTACT;
         if (ANS = 'Y') then begin
           gotoxy(0,16);
           writeln('Remember to delete any parameter sets');
           write(' if necessary');
           write('(CR) to continue'); readln(ANS);
           NEXT3^.NEXT:=NEXT3^.NEXT^.NEXT; end
         else begin NEXT3:=NEXT3^.NEXT; CHNGACT; end;
      end; (* of else *)
end;

begin    end.
```

```
(**********************************************
* This unit is linked to QGERTNET.  It allows   *
* the user to change/delete parameter sets.     *
*       Written by : Anderson & Commeford       *
**********************************************)

(*$S+*)
unit EDITPAR; (* saved as UEDITPAR.TEXT *)

interface

uses GLOBAL;

procedure FINDPAR;

implementation

(**************  CHNGPAR  ********************
* Allows changes of certain fields to parameter *
* sets.         Called by : FINDPAR             *
**********************************************)

procedure CHNGPAR;
begin
   writeln; write('To change a field above, type in the ');
   writeln('first letter ');
   writeln('or number of the field you want to change.');
   writeln('A blank field is the default value.');
   writeln('You may type a Q when you are done with this ');
   write('parameter set.');
   while (ANS <> 'Q') do begin
      repeat
         gotoxy(30,14); write('                              ');
         gotoxy(20,16); write('                              ');
         gotoxy(0,14); write('Enter your choice ');
         write('(P,1,2,3,4,C,Q) ---> ');
         read(ANS);
      until ((ANS='P')or(ANS='1')or(ANS='2')or(ANS='3')
      or(ANS='4')or(ANS='C')or(ANS='Q'));
      if (ANS='Q') then exit(CHNGPAR);
      gotoxy(0,16); write('Enter new value ---> ');
      case ANS of
         'P': readln(NEXT4^.PARAM);
         '1': readln(NEXT4^.PAR1);
         '2': readln(NEXT4^.PAR2);
         '3': readln(NEXT4^.PAR3);
         '4': readln(NEXT4^.PAR4);
         'C': readln(NEXT4^.COMMENT);
      end; (* of case stmt *)
      gotoxy(0,2);write('                                       ');
      write('                              ');
```

```
        gotoxy(3,2); write(NEXT4^.TIPE);
        gotoxy(9,2); write(NEXT4^.PARAM);
        gotoxy(22,2); write(NEXT4^.PAR1);
        gotoxy(30,2); write(NEXT4^.PAR2);
        gotoxy(38,2); write(NEXT4^.PAR3);
        gotoxy(46,2); write(NEXT4^.PAR4);
        gotoxy(3,4); write(NEXT4^.COMMENT);
     end;  (* of while *)
  end;

  (**************  GOTPAR  *********************
  * After finding selected parameter, displays    *
  * information of that parameter set and asks if *
  * deletion is desired.                          *
  *       Called by : FINDPAR                     *
  ************************************************)

  procedure GOTPAR;
  begin
     write(chr(12));
     writeln('Here is your parameter set');
     write('  TYPE  P(ARAM SET#  PAR(1)  PAR(2)  PAR(3)  ');
     write('PAR(4)');
     gotoxy(0,3); write('  C(OMMENT');
     if FINDER then
       begin
        gotoxy(3,2); write(NEXT4^.TIPE);
        gotoxy(9,2); write(NEXT4^.PARAM);
        gotoxy(22,2); write(NEXT4^.PAR1);
        gotoxy(30,2); write(NEXT4^.PAR2);
        gotoxy(38,2); write(NEXT4^.PAR3);
        gotoxy(46,2); write(NEXT4^.PAR4);
        gotoxy(3,4); write(NEXT4^.COMMENT);
       end else
       begin
        gotoxy(3,2); write(NEXT4^.NEXT^.TIPE);
        gotoxy(9,2); write(NEXT4^.NEXT^.PARAM);
        gotoxy(22,2); write(NEXT4^.NEXT^.PAR1);
        gotoxy(30,2); write(NEXT4^.NEXT^.PAR2);
        gotoxy(38,2); write(NEXT4^.NEXT^.PAR3);
        gotoxy(46,2); write(NEXT4^.NEXT^.PAR4);
        gotoxy(3,4); writ[illegible]T4^.NEXT^.COMMENT);
       end;
     repeat
       gotoxy(57,7); wri'e [illegible]
       gotoxy(0,7);
       write('Do you want to delete this entire parameter ');
       write('set? Y/N ---> ');
       read(ANS);
     until ((ANS='Y') or (ANS='N'));
  end;
```

```
(**************  FINDPAR  *********************
* Finds the selected parameter set and then     *
* either deletes the set or calls CHNGPAR.      *
*       Called by : unit EDIT                   *
***********************************************)

procedure FINDPAR;
begin
   if (BASE4 = nil) then
      begin writeln('Can"t find ',STRSELEC,ELEMENT);
         writeln('<CR> to continue'); read(ANS);
         exit(FINDPAR);
      end; (* of then *)
   FINDER:=true;
   if (NEXT4^.PARAM = ELEMENT) then
      begin GOTPAR;
          if (ANS = 'Y') then BASE4:=BASE4^.NEXT
            else CHNGPAR;
      end
      else begin
         FINDER:=false;
         repeat
            if (NEXT4^.NEXT^.PARAM <> ELEMENT) then
            NEXT4:=NEXT4^.NEXT;
         until ((NEXT4^.NEXT=nil) or (NEXT4^.NEXT^.PARAM=ELEMENT));
         if (NEXT4^.NEXT=nil) then
            begin writeln('Can"t find ',STRSELEC,ELEMENT);
               writeln('<CR> to continue'); read(ANS);
               exit(FINDPAR);
            end; (* of then *)
         GOTPAR;
         if (ANS = 'Y') then
           NEXT4^.NEXT:=NEXT4^.NEXT^.NEXT
         else begin NEXT4:=NEXT4^.NEXT; CHNGPAR; end;
      end; (* of else *)
end;

begin    end.
```

```
(**********************************************
* This is the main editor unit and is linked to *
* QGERTNET.  It presents a main menu which asks *
* the user what kind of symbol needs to be      *
* changed/deleted.  It then calls the appropri- *
* ate unit; i.e. UEDITSR, UEDITSS, ...          *
*       Written by : Anderson & Commeford       *
**********************************************)

(*$S+*)
unit EDIT;  (* saved as UEDIT.TEXT *)

interface

uses TURTLEGR, GLOBAL, RITE,
   (*$U #4:UINN.CODE *) INN,
   (*$U #4:UOUT.CODE *) OUT,
   (*$U #4:UEDITSR.CODE *) EDITSR,
   (*$U #4:UEDITSS.CODE *) EDITSS,
   (*$U #5:UEDITQUE.CODE*) EDITQUE,
   (*$U #4:UEDITACT.CODE *) EDITACT,
   (*$U #4:UEDITPAR.CODE *) EDITPAR;

procedure EDITOR;

implementation

(**************  OTHERWISE  ******************
* Depending on what user wants to edit, sets up *
* appropriate link list and calls FIND procedure*
*       Called by : EDITOR                      *
**********************************************)

procedure OTHERWISE;
begin
   case SELEC of
      'B','C': begin NEXT1:=BASE1;
                 STRSELEC:='node number'; end;
      'D','F': begin NEXT2:=BASE2;
                 STRSELEC:='node number'; end;
      'E'    : begin NEXT3:=BASE3;
                 STRSELEC:='activity number'; end;
      'G'    : begin NEXT4:=BASE4;
                 STRSELEC:='parameter set number'; end;
      'H'    : begin NEXT5:=BASE5;
                 STRSELEC:='node number'; end;
   end; (* of case stmt *)
   if (SELEC <> 'I') then begin
     writeln; write('Edit which ',STRSELEC,' ---> ');
     readln(ELEMENT);
   end;  (* of then *)
   SAFETY:=false;
   case SELEC of
      'B','C': FINDSOUREG;
      'D','F': FINDSTASIN;
      'E'  : FINDACT;
      'G'  : FINDPAR;
      'H'  : FINDQUE;
      'I'  : begin UPDATE:=true; SAVESCREEN; INITTURTLE;
               LOADSCREEN; UPDATE:=false;
             end;
   end;  (* of case *)
end;

(***************  LISTER  **********************
* Lists the main menu of the editor and gets the*
* answer.        Called by : EDITOR            *
************************************************)

procedure LISTER;
begin
   write(chr(12)); writeln; writeln;
   writeln('Which of the following do you want to edit:');
   writeln('   A) Quit the editor');
   writeln('   B) Source node');
   writeln('   C) Regular node');
   writeln('   D) Statistics node');
   writeln('   E) Activity');
   writeln('   F) Sink node');
   writeln('   G) Parameter card');
   writeln('   H) Queue node');
   writeln('   I) Update - to same filename');
   repeat
      writeln; write('Enter your choice ---> ');
      read(selec);
   until ((SELEC > chr(64)) and (SELEC < chr(74)));
end;

(***************  EDITOR  **********************
* Main proc of the editor.  Stays in editor    *
* until LISTER returns an A, then before leaving*
* makes sure that edited network has been saved.*
*       Called by : MENU                       *
************************************************)

procedure EDITOR;
(*$N+*)
begin
   write(chr(12));
   if (SAFETY=false) then
      begin writeln('SAVE the network first!!');
         writeln('<CR> to continue'); read(ANS);
         exit(EDITOR);
      end;
    if (length(FILENAME)=0) then
      begin writeln('LOAD your network first!!');
        writeln('(CR) to continue'); read(ANS);
        exit(EDITOR);
      end;
    SELEC:='B';
    while (SELEC <> 'A') do begin
    LISTER;
    if (SELEC = 'A') then
      begin if (not SAFETY) then SAVESCREEN;
      exit(EDITOR); end;
    OTHERWISE;
    end; (* of while *)
end;

begin     end.
```

```
(*********************************************************
* Calls the initialization routines, loads shapes, loops *
* in MYPLOT till the EXT routine is called.  As MYPLOT   *
* draws the network, a data structure (made up of link   *
* lists) is built from the information supplied by the   *
* user in xxxINFO procedures.  This data is used by the  *
* program CODEGEN to generate QGERT source code.  From   *
* this program networks can be SAVEd, LOADed, created,   *
* and EDITed.  Also, the program CODEGEN can be executed *
* to generate QGERT source code of a network.            *
*       Written by : Anderson & Commeford                *
* Modification of a program written by Dan Sokol.        *
*********************************************************)

program QGERTNET;
(*$S++*)

uses PEEKPOKE,TURTLEGR,CHAINSTUFF,GLOBAL,RITE,NODEINFO,
     ACTPAR,
     (*$U #4:UINN.CODE*) INN,
     (*$U #4:UOUT.CODE*) OUT,
     (*$U #4:UEDITSR.CODE*) EDITSR,
     (*$U #4:UEDITSS.CODE*) EDITSS,
     (*$U #5:UEDITQUE.CODE*) EDITQUE,
     (*$U #4:UEDITACT.CODE*) EDITACT,
     (*$U #4:UEDITPAR.CODE*) EDITPAR,
     (*$U #5:UEDIT.CODE*) EDIT;

var   HEAP  : ^integer;

(******************** KEY **********************************
* Replaces applestuff KEYPRESS function which doesn't work  *
* if there is a card in slot #3.  Called from: GETXY        *
***********************************************************)
function KEY : boolean;
        var CLEAR,KEYBOARD,TEMP : integer;
        begin
           CLEAR:=-16368;KEYBOARD:=-16384;
           TEMP:=PEEK(KEYBOARD);
           if TEMP > 128 then
           begin KEY:=true; POKE(CLEAR,TEMP); end
           else KEY:=false;
        end;

(***************  SETUPAD & READPAD  *****************
*    Assembly language procedures to setup and        *
*    read the Graphics Tablet.                        *
*    Called by : GETXY and Main program loop          *
*****************************************************)
procedure SETUPAD; external;
procedure READPAD; external;
```

```
(************** EXT ***************************************
* The only legitimate exit from QGERTNET                    *
*     Called by: MENU                                       *
************************************************************)
procedure ext;
      begin
        write(chr(12));
        write('Do you want to save the screen?  '); read(CH);
        if (CH='y') or (CH='Y') then SAVESCREEN;
        if SAFETY then exit(PROGRAM);
        begin writeln; writeln;
          write('The screen was NOT saved. ');
          write('Do you want to exit anyway?  ');
          read(CH);
          if (CH='Y') or (CH='y') then exit(program);
        end;
      end;

(************** GETXY *************************************
* Read tablet and get the X & Y coordinates.  Deter-      *
* mine if X & Y are on screen (VALIDXY).                   *
*   Called by: MENU, Main program loop                    *
***********************************************************)
procedure GETXY;
      var B1,B6,B7,B8,B9: integer;
      begin
        B1:=640; B9:=648;
        B6:=645; B7:=646; B8:=647;
        repeat READPAD;
          PEN:=PEEK(B1);
          X:=256*(PEEK(B7))+PEEK(B6);
          Y:=256*(PEEK(B9))+PEEK(B8);
          if KEY then
            begin
              VALIDXY:=false;
              X:=-100; Y:=-100;
              exit(GETXY);
            end;
        until PEN=2;
        if (X>=0) and (X<280) and (Y>=0) and (Y<192) then
          begin VALIDXY:=true; Y:=191-Y; end
          else VALIDXY:=false;
      end;

(*********** PRINTYPE ********************************
* Prints out the name of the device that will be     *
* plotted.  Determined by the value of D.            *
*     Called by: LISTALL, GETYPE                     *
******************************************************)
procedure PRINTYPE;
      begin
```

```
        case D of
          0:IDENT:='** INVALID **';
          1:IDENT:='SOURCE NODE';
          2:IDENT:='REGULAR NODE';
          3:IDENT:='STATISTICS NODE';
          4:IDENT:='ACTIVITY';
          5:IDENT:='QUE NODE';
          6:IDENT:='SINK NODE';
        end; (* of case stmt *)
        write(IDENT);
      end;

(*********************** LISTALL ******************
*   List all the names of all the devices that      *
*   can be plotted on the text screen.              *
*   Called by : MENU                                *
***************************************************)
procedure LISTALL;
var   I  : integer;
        begin
          write(chr(12));
          for I:=1 to 6  do
             begin write(I,' - '); D:=I; PRINTYPE; writeln;
             end;
          gotoxy(0,22); write('<CR> to continue'); read(CH);
        end;

(********** FORWARD REFRENCES ********)
procedure LISTMODE; forward;
procedure MENU; forward;
procedure CANCEL; forward;

(***************   GETYPE   ***************************
*    Sets up text display to show what is being        *
*    plotted and the status of the X & Y locks.        *
*    Called by : MENU                                  *
*****************************************************)
procedure GETYPE;
        begin
          write(chr(12)); gotoxy (0,19);
          write ('Device type >> '); PRINTYPE; CH:='P';
          LISTMODE;
        end;

(**************   MYPLOT   *********************
*  Plots a device if X & Y are valid,           *
*  calls MENU if not.                           *
*  Called by : Main program loop.               *
***********************************************)
procedure MYPLOT;
      begin
      (*$R TURTLEGR*)
```

```
          pencolor (white);
          if not VALIDXY then begin write(chr(7)); MENU;end;
          if D=0 then exit (MYPLOT);
          if (D = 4) then VALIDXY := true;
          if LOCKY then X:=LASTX;
          if LOCKX then Y:=LASTY;
          if VALIDXY then
             begin SAFETY:=false;
                case D of
                   1: begin drawblock(ARROW,4,0,0,21,21,X,Y,DMODE);
                      drawblock(NODEL,4,0,0,21,21,X+11,Y,DMODE);
                      drawblock(NODER,4,0,0,21,21,X+32,Y,DMODE);
                      SOUREGINFO;
                      end;
                   2: begin drawblock(NODEL,4,0,0,21,21,X,Y,DMODE);
                      drawblock(NODER,4,0,0,21,21,X+21,Y,DMODE);
                      SOUREGINFO;
                      end;
                   3: begin drawblock(NODEL,4,0,0,21,21,X,Y,DMODE);
                      drawblock(NODER,4,0,0,21,21,X+21,Y,DMODE);
                      STASININFO;
                      end;
                   4: ACTINFO;
                   5: begin drawblock(QNODEL,4,0,0,21,21,X,Y,DMODE);
                      drawblock(QNODER,4,0,0,21,21,X+21,Y,DMODE);
                      QUEINFO;
                      end;
                   6: begin drawblock(NODEL,4,0,0,21,21,X,Y,DMODE);
                      drawblock(NODER,4,0,0,21,21,X+21,Y,DMODE);
                      drawblock(ARROW,4,0,0,21,21,X+42,Y,DMODE);
                      STASININFO;
                      end;
                end;  (* of case statement *)
             CANCEL;LISTMODE;
             end;
             DMODE:=14;
        end;

(******************* LISTMODE ************************
*   More info for the text screen.                      *
*   Called by : MENU,GETYPE,Main program loop           *
*******************************************************)
procedure LISTMODE;
     begin
       gotoxy(0,15); write('Mode = ');
       case CH of
       'P': begin write('Plot devices'); gotoxy(17,5);
              if LOCKY then
              begin write('<<< X AXIS IS LOCKED AT ');
                write(LASTY,' >>>');
              end;
              if LOCKX then
```

```
                begin write('<<<< Y AXIS IS LOCKED AT ');
                  write(LASTX,' >>>');
                end;
                if (not LOCKX)  and (not LOCKY) then write
                ('                                              ');
              end;
          'B','C','D': write('setup lock');
          'Z': write(' ????????? ');
          end;  (* of case stmt *)
          gotoxy(17,5);
          if (not LOCKX)  and (not LOCKY) then write
          ('                                              ');
        end;

(**************** CLEARSCREEN ******************
* Clears Hires screen 1.    Called by: MENU     *
************************************************)
procedure CLEARSCREEN;
      begin
        write(chr(12));
        write('Clear the screen - Are you sure? ');
        read(CH); if (CH='Y') or (CH='y') then
           begin initturtle; release(HEAP); BASE1:=nil;
              BASE2:=nil; BASE3:=nil; BASE4:=nil;
              BASE5:=nil; FILENAME:='';
           end;
      end;

(********** CANCEL ******************************
* Fixes text screen on leaving any command.     *
*     Called by: MENU, MYPLOT, Main program     *
************************************************)
procedure CANCEL;
      begin
        write(chr(12));
        gotoxy(27,12); write('*** NO MODE ACTIVE ***');
        write(chr(7));D:=0; CH:='Z';
      end;

(***************  SETLOCK  *************************
* Locks the x or y coordinate; or clears the lock.  *
*       Called by : MENU                            *
****************************************************)

procedure SETLOCK;
begin
  case CH of
  'C': begin  writeln; writeln;
         write('Use pen to select row');
         repeat GETXY; until VALIDXY; LASTX:=X; LASTY:=Y;
         LOCKX:=true; LOCKY:=false; end;
  'D': begin  writeln; writeln;
```

```
             write('Use pen to select column');
             repeat GETXY; until VALIDXY; LASTX:=X; LASTY:=Y;
             LOCKX:=false; LOCKY:=true; end;
   'B': begin  LOCKX:=false; LOCKY:=false; end;
    end;  (* of case *)
end;

(*******  MENU  ***********************
* Mode selection happens here.        *
*     Called by: MYPLOT               *
**************************************)
procedure MENU;
      const  STR='CODEGEN';
      var XPOS, YPOS : integer;
      begin
        D:=0;
        (* actual value of the divisor may *)
        (* vary from tablet to tablet      *)
        XPOS:=trunc((X+65)/16.0);
        YPOS:=trunc((Y-224)/16.0);
        case YPOS of
          3: case XPOS of   (* Bottom row, left to right *)
            0: begin CLEARSCREEN; CANCEL;LISTMODE;end;
            1: begin SETCHAIN(STR);EXT;end;
            2: begin LOADSCREEN; CANCEL;LISTMODE; end;
            3: begin SAVESCREEN; CANCEL;LISTMODE; end;
            4: EXT;
            5: begin LISTALL; CANCEL; LISTMODE; end;
            6: begin EDITOR; CANCEL; LISTMODE; end;
            7: begin CH:='B'; LISTMODE;
                 SETLOCK; CANCEL; LISTMODE; end;
            8: begin CH:='D'; LISTMODE;
                 SETLOCK; CANCEL; LISTMODE; end;
            9: begin CH:='C'; LISTMODE;
                 SETLOCK; CANCEL; LISTMODE; end;
            10: begin write(chr(7)); gotoxy(0,22);
                  write('memory available is ',memavail);
                end;
          end;   (* of YPOS=3 *)
          (* 2nd row from the bottom *)
          2: begin D:=XPOS+1;
               if D>6 then D:=0; GETYPE;
             end;
          (* 3rd row from the bottom *)
          1: begin D:=XPOS+20;
               if D>20 then D:=0; GETYPE;
             end;
        end;   (* of YPOS case stmt *)
      end;
```

```
(**************** MAIN PROGRAM LOOP ********************)

begin
(*$N+*)
(*$R PEEKPOKE,READPAD*)
  (* initialize booleans *)
  SAFETY:=true;
  HELLFREEZESOVER:=false;
  UPDATE:=false;
  BASE1:=nil;
  BASE2:=nil;
  BASE3:=nil;
  BASE4:=nil;
  BASE5:=nil;
  FILENAME:='';
  INVERSE := false;
  LOCKX:= false;
  LOCKY:= false;
  (* initialize plotting mode *)
  write(chr(7));write(chr(7));write(chr(7));
  DMODE:=14;write(chr(12));
  gotoxy(25,12); write('loading the QGERT symbols');
  GETSHAPES;
  (* setup pad and screen *)
  SETUPAD; INITTURTLE;
  (* setup text screen *)
  CANCEL; LISTMODE; mark(HEAP);
  (* let's doit *)
  repeat
    GETXY;
    MYPLOT;
  until HELLFREEZESOVER;
end.
```

Appendix E
Program CODEGEN

```
(*********************************************
* This program is the same as the unit UINN    *
* except that it does not have any graphics.   *
* It was not considered necessary to have      *
* graphics to generate source code.  It loads  *
* the data structure of a SAVEd network into   *
* link lists.                                  *
*       Called by : CODEGEN                    *
*       Written by : Anderson & Commeford      *
*********************************************)

(*$S+*)
unit LOADER; (* saved as ULOADER.TEXT *)

interface

uses GLOBAL;

procedure INDATA(var SRFF: SOUREGFIL;
                 var SSFF: STASINFIL;
                 var AFF : ACTFIL;
                 var PFF : PARFIL;
                 var QFF : QUEFIL);

procedure LOADSCREEN;

implementation

procedure INDATA;
var   DUMMY      : string;
      AGAIN      : boolean;
      ANS        : char;

begin write(chr(12));
   writeln('          GENERATING QGERT SOURCE CODE');
   writeln;
   write('Generate source code for what file ---> ');
   readln(FILENAME);
   if length(FILENAME) =  0 then exit(INDATA);
   if length(FILENAME) > 10 then
      begin writeln; writeln('Filename too long!!',chr(7));
         exit(INDATA);
      end;

   DUMMY := concat(FILENAME,'.SOUREG');
```

```
(*$I-*)
   reset(SRFF,DUMMY);
   if (IORESULT <> 0)
      then begin writeln;
         writeln('File called ',DUMMY,' not found');
         writeln('<CR> to continue'); read(ANS);
(*$I-*)
         FINDER:=false;exit(INDATA);end;
   writeln;writeln('Reading ',DUMMY,' from disk');
   BASE1:=nil;
   while not eof(SRFF) do begin
      new(NEXT1);
      NEXT1^ := SRFF^;
      NEXT1^.NEXT:=BASE1; BASE1:=NEXT1;
      get(SRFF);
   end;
   close(SRFF);

   DUMMY := concat(FILENAME,'.STASIN');
   writeln;writeln('Reading ',DUMMY,' from disk');
   reset(SSFF,DUMMY); BASE2:=nil;
   while not eof(SSFF) do begin
      new(NEXT2);
      NEXT2^ := SSFF^;
      NEXT2^.NEXT:=BASE2; BASE2:=NEXT2;
      get(SSFF);
   end;
   close(SSFF);

   DUMMY := concat(FILENAME,'.QUE');
   writeln;writeln('Reading ',DUMMY,' from disk');
   reset(QFF,DUMMY); BASE5:=nil;
   while not eof(QFF) do begin
      new(NEXT5);
      NEXT5^ := QFF^;
      NEXT5^.NEXT:=BASE5; BASE5:=NEXT5;
      get(QFF);
   end;
   close(QFF);

   DUMMY := concat(FILENAME,'.ACT');
   writeln;writeln('Reading ',DUMMY,' from disk');
   reset(AFF,DUMMY); BASE3:=nil;
   while not eof(AFF) do begin
      new(NEXT3);
      NEXT3^ := AFF^;
      NEXT3^.NEXT:=BASE3; BASE3:=NEXT3;
      get(AFF);
   end;
   close(AFF);
```

```
    DUMMY := concat(FILENAME,'.PAR');
    writeln;writeln('Reading ',DUMMY,' from disk');
    reset(PFF,DUMMY); BASE4:=nil;
    while not eof(PFF) do begin
       new(NEXT4);
       NEXT4^ := PFF^;
       NEXT4^.NEXT := BASE4; BASE4 := NEXT4;
       get(PFF);
    end;
    close(PFF);

end;

(*************** LOADSCREEN ******************)

procedure LOADSCREEN;
   begin INDATA(SRF,SSF,AF,PF,QF); end;



begin end.
```

```
(***************  CODEGEN  *********************
* This program first LOADs a SAVEd network, then*
* builds and writes a General card to the source*
* code text file.  The source code for Source   *
* nodes follows.  Then source code for Regular, *
* Stat, and Queue nodes; Activities; and Par-   *
* ameter Sets.  After the complete source code  *
* is written and reviewed, QGERTNET is automat- *
* ically executed with the chain command.       *
*       Called by : MENU of QGERTNET            *
*       Written by : Anderson & Commeford       *
************************************************)

(*$S++*)
program CODEGEN;

uses CHAINSTUFF, GLOBAL,
     (*$U #5:ULOADER.CODE*) LOADER;

const STRNG='QGERTNET';         (* used in chain command *)
type LINKA = ^NODECODE;
     NODECODE = record
       NEXT     : LINKA;
       CODE     : string[72];
     end;
     LINKB = ^SINKCODE;
     SINKCODE = record
       NEXT     : LINKB;
       CODE     : string[72];
     end;

var  NEXTA,BASEA  : LINKA;     (* stores the source code *)
     NEXTB,BASEB  : LINKB;     (* stores the source code *)
     DIFF,K       : integer;   (* used to tab comment field *)
     WORKST       : string;    (* used to generate code  *)
     ST1,ST2      : string[2]; (* counts stat/sink nodes *)
     SOURCECODE   : text;      (* text file for source code *)

(***************  STATSEARCH  ******************
* Searches for and counts stat and sink nodes   *
* in the link list.                             *
*       Called by : GENERALCARD                 *
************************************************)

procedure STATSEARCH;
var   M,N    : integer;

begin
   NEXT2:=BASE2; M:=0; N:=0;
   BASEA:=nil; BASEB:=nil;
   while (NEXT2 <> nil) do begin
```

```
        WORKST:=concat(NEXT2^.TIPE,',',NEXT2^.NODENUM,',');
        WORKST:=concat(WORKST,NEXT2^.INITIAL,',');
        WORKST:=concat(WORKST,NEXT2^.SUBSEQUENT,',');
        WORKST:=concat(WORKST,NEXT2^.BRANCHING,',');
        WORKST:=concat(WORKST,NEXT2^.STAT,',');
        WORKST:=concat(WORKST,NEXT2^.UPPER,',');
        WORKST:=concat(WORKST,NEXT2^.WIDTH,',');
        WORKST:=concat(WORKST,NEXT2^.CHOICE,'*');
        DIFF:=30 - length(WORKST);
        for K:=1 to DIFF do
          begin  WORKST:=concat(WORKST,' ');  end;
        WORKST:=concat(WORKST,NEXT2^.COMMENT);
        if (NEXT2^.TIPE = 'STA')
           then begin M:=M+1; new(NEXTA);
             NEXTA^.CODE:=WORKST;
             NEXTA^.NEXT:=BASEA; BASEA:=NEXTA;
           end
           else begin N:=N+1; new(NEXTB);
             NEXTB^.CODE:=WORKST;
             NEXTB^.NEXT:=BASEB; BASEB:=NEXTB;
           end;
        NEXT2:=NEXT2^.NEXT;
     end;   (* of while *)
     str(M,ST1); str(N,ST2);
  end;

  (**************  GENERALCARD  ****************
  * Builds the General card and writes it to the  *
  * source code text file.                        *
  *       Called by : CODEGEN                     *
  ***********************************************)

  procedure GENERALCARD;

  begin
     write(chr(12));
     writeln('           GENERATING QGERT SOURCE CODE');
     writeln('            Memory available is ',memavail);
     writeln;write('Enter your name --->  '); readln(WORKST);
     ST:=concat('GEN,',WORKST);
     write('Enter project name --->  ');
     readln(WORKST);
     ST:=concat(ST,',',WORKST);
     write('Enter date as MM,DD,YYYY --->  ');
     readln(WORKST);
     ST:=concat(ST,',',WORKST);
     STATSEARCH;
     ST:=concat(ST,',',ST1,',',ST2);
     write('Enter number of sink node releases to ');
     write('end a run --->  ');
     readln(WORKST);
     ST:=concat(ST,',',WORKST);
```

```
    write('Enter time to end one run of the network ');
    write('---> ');
    readln(WORKST);
    ST:=concat(ST,',',WORKST);
    write('Enter number of runs of the network --->  ');
    readln(WORKST);
    ST:=concat(ST,',',WORKST);
    write('Enter type of output reports desired ');
    write('(F,E,C or S) --->  ');
    readln(WORKST);
    ST:=concat(ST,',',WORKST,'*');
    writeln(SOURCECODE,ST);
end;

(***************  SOURCESEARCH  ****************
* Searches for Source nodes and immediately    *
* writes them to source code text file.  Also  *
* finds the Regular nodes, builds their source *
* code and writes it to the source code text   *
* file after all Source nodes are written.     *
*        Called by : CODEGEN                   *
***********************************************)

procedure SOURCESEARCH;

begin
   NEXT1:=BASE1;
   while (NEXT1 <> nil) do begin
     WORKST:=concat(NEXT1^.TIPE,',',NEXT1^.NODENUM,',');
     WORKST:=concat(WORKST,NEXT1^.INITIAL,',');
     WORKST:=concat(WORKST,NEXT1^.SUBSEQUENT,',');
     WORKST:=concat(WORKST,NEXT1^.BRANCHING,',');
     WORKST:=concat(WORKST,NEXT1^.MARK,',');
     WORKST:=concat(WORKST,NEXT1^.CHOICE,'*');
     DIFF:=30 - length(WORKST);
     for K:=1 to DIFF do
       begin  WORKST:=concat(WORKST,' ');  end;
     WORKST:=concat(WORKST,NEXT1^.COMMENT);
     if (NEXT1^.TIPE='SOU')
        then begin writeln(SOURCECODE,WORKST); end
        else begin new(NEXTA);
           NEXTA^.CODE:=WORKST;
           NEXTA^.NEXT:=BASEA; BASEA:=NEXTA;
         end;
     NEXT1:=NEXT1^.NEXT;
   end;   (* of while *)
end;
```

```
(**************  QUESEARCH  *****************
* Searches for Queue nodes, builds source code, *
* and immediately writes to the source code     *
* text file.           Called by : CODEGEN      *
*********************************************)

procedure QUESEARCH;

begin
   NEXT5:=BASE5;
   while (NEXT5 <> nil) do begin
     WORKST:=concat(NEXT5^.TIPE,',',NEXT5^.NODENUM,',');
     WORKST:=concat(WORKST,NEXT5^.INITIAL,',');
     WORKST:=concat(WORKST,NEXT5^.CAPACITY,',');
     WORKST:=concat(WORKST,NEXT5^.BRANCHING,',');
     WORKST:=concat(WORKST,NEXT5^.RANKING,',');
     WORKST:=concat(WORKST,NEXT5^.BALKERS,',');
     WORKST:=concat(WORKST,NEXT5^.UPPER,',');
     WORKST:=concat(WORKST,NEXT5^.WIDTH,'*');
     DIFF:=30 - length(WORKST);
     for K:=1 to DIFF do
       begin  WORKST:=concat(WORKST,' ');  end;
     WORKST:=concat(WORKST,NEXT5^.COMMENT);
     writeln(SOURCECODE,WORKST);
     NEXT5:=NEXT5^.NEXT;
   end;   (* of while *)
end;
(**************  ACTIVSEARCH  ****************
* Builds the source code for Activities, and    *
* writes it to the source code text file.       *
*       Called by : CODEGEN                     *
*********************************************)

procedure ACTIVSEARCH;

begin
   NEXT3:=BASE3;
   while (NEXT3 <> nil) do begin
      WORKST:=concat(NEXT3^.TIPE,',',NEXT3^.START,',');
      WORKST:=concat(WORKST,NEXT3^.IND,',');
      WORKST:=concat(WORKST,NEXT3^.DISTR,',');
      WORKST:=concat(WORKST,NEXT3^.PARAM,',');
      WORKST:=concat(WORKST,NEXT3^.ACTNUM,',');
      WORKST:=concat(WORKST,NEXT3^.SERVERS,'*');
      DIFF:=30 - length(WORKST);
      for K:=1 to DIFF do
        begin  WORKST:=concat(WORKST,' ');  end;
      WORKST:=concat(WORKST,NEXT3^.COMMENT);
      writeln(SOURCECODE,WORKST);
      NEXT3:=NEXT3^.NEXT;
   end;   (* of while *)
end;
```

```
(***************  PARSEARCH  *******************
* Builds the source code for the Parameter Sets *
* and writes it to the source code text file.   *
*       Called by : CODEGEN                     *
************************************************)

procedure PARSEARCH;

begin
   NEXT4:=BASE4;
   while (NEXT4 <> nil) do begin
      WORKST:=concat(NEXT4^.TIPE,',',NEXT4^.PARAM,',');
      WORKST:=concat(WORKST,NEXT4^.PAR1);
      if (length(NEXT4^.PAR4)=0) and (length(NEXT4^.PAR3)=0)
         then begin if (length(NEXT4^.PAR2)<>0)
            then WORKST:=concat(WORKST,',',NEXT4^.PAR2); end
         else begin WORKST:=concat(WORKST,',',NEXT4^.PAR2);
            WORKST:=concat(WORKST,',',NEXT4^.PAR3);
            if (length(NEXT4^.PAR4)<>0)
               then WORKST:=concat(WORKST,',',NEXT4^.PAR4);
         end; (* of else *)
      WORKST:=concat(WORKST,'*');
      DIFF:=30 - length(WORKST);
      for K:=1 to DIFF do
        begin  WORKST:=concat(WORKST,' ');  end;
      WORKST:=concat(WORKST,NEXT4^.COMMENT);
      writeln(SOURCECODE,WORKST);
      NEXT4:=NEXT4^.NEXT;
   end;   (* of while *)
end;

(***************  PRINTCODE  *******************
* After the source code is completed and written*
* to the text file, displays the source code on *
* the screen for review.                        *
*       Called by : CODEGEN                     *
************************************************)

procedure PRINTCODE;
begin
   reset(SOURCECODE);write(chr(12));
   gotoxy(10,1);
   writeln('This is what was written to ',DUMMY);
   writeln;
   K:=3;
   repeat
      K:=K + 1;
      if (K > 20) then
      begin  gotoxy(0,23); write('<CR> to see some more');
         read(CH); K:=1; writeln;
      end; (* of then *)
```

```
         readln(SOURCECODE,WORKST);writeln(WORKST);
      until (WORKST = 'FINISH*');
      writeln; write('<CR> to continue ');
      read(CH);
   end;

   (*****    MAIN PROGRAM    ******)

   begin
   (*$N+*)
      write(chr(12));
      write(chr(7)); write(chr(7)); write(chr(7));
      FINDER:=true;LOADSCREEN;
      if not FINDER
         then begin setchain(STRNG); exit(PROGRAM);end;
      DUMMY:=concat(FILENAME,'.TEXT');
      rewrite(SOURCECODE,DUMMY);
      GENERALCARD;
      SOURCESEARCH;
      while (NEXTA <> nil) do
      begin   writeln(SOURCECODE,NEXTA^.CODE);
        NEXTA:=NEXTA^.NEXT;
      end;   (* of while *)
      QUESEARCH;
      ACTIVSEARCH;PARSEARCH;
      while (NEXTB <> nil) do
      begin   writeln(SOURCECODE,NEXTB^.CODE);
        NEXTB:=NEXTB^.NEXT;
      end;   (* of while *)
      writeln(SOURCECODE,'FINISH*'); PRINTCODE;
      close(SOURCECODE,LOCK);setchain(STRNG);
   end.
```

Appendix F

USER'S MANUAL

AUTOMATED QGERT SOURCE CODE GENERATION

BY

CAPT. G. ANDERSON AND CAPT. C. COMMEFORD

Table of Contents

Introduction

This is the operation manual for the Apple Graphics Tablet as used with QGERTNET. The Tablet itself is a hands-on product, meaning the best way to learn how to use it is to experiment with it. Of course, this manual is designed to make that experimentation less painful. Therefore, the best results will occur if as you read about the capabilities in this manual, you also attempt them on the tablet.

The first chapter describes how to set up your Tablet. It closely follows Chapter 1 of the operation and reference manual of the Graphics Tablet. The difference being that this manual is written specifically for the Pascal program QGERTNET, not for Applesoft programs. Chapter 2 describes the various Pascal procedures used by the tablet. They allow you to draw QGERT networks on a high-resolution graphics screen, input information on a text screen, edit existing networks, and generate QGERT source code. You don't need to know too much about the Apple computer or Pascal to run this program. You will have to know QGERT. This manual is not intended to be a text on QGERT, although it could be useful to someone learning QGERT in class or from a text book.

Getting Started

What You Will Need

In order to use this software package you will need the following:

1) An Apple II+, 64K RAM
2) Sup'R' Term 80 column board in slot #3
3) Graphics Tablet board in slot #5
4) Two monitors
5) Two disk drives
6) Modem

The above configuration is the recommended one. Minor modifications will work at a degraded level. For instance, only one disk drive is necessary to run the program. However, making back-up copies of your disks is extremely difficult with only one drive. Also, one monitor will work, but you find yourself switching constantly from 40 columns to 80 columns and vice versa. This can get confusing and tiring. The majority of this software package was developed with only one screen, but a toggle switch was used to go from screen to screen. Finally, the Apple IIe should work with this package with no degradation at all.

Initial Setup

If your graphics tablet is not hooked up to your Apple ye then refer to the Graphics Tablet manual, pages 5-12. Onc the tablet is connected to your Apple and aligned, you ne to set up your own menu rows. This can be done with grease pencil. The last two rows on the tablet should lo like:

SOU	REG	STA	ACT	QUE	SIN				
CLR	GEN	LOAD	SAVE	EXT	LIST	EDIT	CLR LOCKS	LOCK X	LOCK Y

The following is a brief explanation of the abo commands/symbols.

SOU	- Source node
REG	- Regular node
STA	- Statistics node
ACT	- Activity
QUE	- Queue node
SIN	- Sink node
CLR	- Clear the screen
GEN	- Generate Q-GERT source code

LOAD	- Load network from disk
SAVE	- Save network to disk
EXT	- Exit the program
LIST	- Lists the available Q-GERT symbols
EDIT	- Invokes the editor
CLR LOCKS	- Clears any coordinate that has been locked
LOCK X	- Locks the x coordinate
LOCK Y	- Locks the y coordinate

You are now ready to use QGERTNET to help you with QGERT simulation.

Using the Software Package

Boot-Up

To start the program, simply put the disk called "AUTO" in disk drive #1 and turn on the computer. Soon you see a title page with the name of the software package, authors, etc. At the bottom of the screen you will see a question asking whether or not you want instructions. These are just a condensed version of this manual. If you ever need a quick review, just type "Y" after the question. After the review (or if you typed "N" to the review question) the main program automatically starts execution.

Main Program

The main program is called QGERTNET. Through this program all actions are accessible. Along the bottom two rows of the tablet you should see the menu (if you don't, go back to Initial Setup of the first chapter). By placing the graphics pen in one of these blocks, you can select one of many commands or QGERT symbols.

To select a command/symbol, touch the point of the graphic's pen anywhere inside the corresponding square and press down.

Hold the pen down until you hear a beep. If you don't hear a beep, check the text screen to see if your selction was activated anyway. If it wasn't try depressing the pen again.

The following pages describe each command and symbol. As you read about each command or symbol, locate it on the two bottom rows of the graphics tablet.

CLR

This is the first of the commands we will describe. It allows you to clear the graphics screen. This command would be used after you've finished a network and want to start another one. Be careful! This command does ask you if you're sure, but once you clear the graphics screen, any network and its data structure that was there is gone. Therefore, always make sure that you SAVE your network before using this command. After using this command the graphics screen will be completely blank, and the text screen will tell you that no mode is active.

LOAD

By selecting this command you can load a network and its data structure which you previously saved. Before you select this command always make sure that your graphics screen is cleared by using CLR. The main use of this command would be so you can bring a network into the Apple's memory and add to it. We will talk about deleting or changing a network later.

NOTE : When you load a network, more than one file will be opened and read. However, every file that is read will have the same prefix which you supply in response to "Load which file". Therefore, the data structure actually comes from a group of related files. In this manual the word file is used to designate this group of related files.

SAVE

This command allows you to save a network and its data structure to disk so that you can work on it again in the future (remember the LOAD function?). You can use this function any time you want or need to. You can save a network several times during one session. However, you must realize that only the last version saved will be in any particular filename. When the text screen asks for a filename, type one in like NET, NET3, or whatever you feel is appropriate. The program does the rest.

EXT

One more command before you get to play. This command is the recommended way to exit QGERTNET. It will make sure you have saved the present network if it was changed before quitting the program.

Exercise #1

By now you are probably anxious to do something. Remember, experimentation is very important in learning to use this tool. So let's use some of the above commands. First, follow the Boot-Up instructions at the beginning of this chapter. Once you get to the question "Do you want instructions?", type "N" since you are reading this manual. Those instructions are only there if you want a quick review of this manual. You now see a statement at the bottom of the screen asking for a carriage return. Press the return key (hereafter denoted by <CR>) and soon you will see the text screen clear and tell you that it is loading LOGIC.CHARSET. This is a file that contains the QGERT symbols. Then you will hear 3 beeps. This denotes the beginning of any program in this software package. And finally you see:

```
NO MODE ACTIVE
```

```
device >>> ????????
```

Whenever you see this on the text screen, it means the computer is waiting for you to select a command or symbol. Now, let's try the LOAD command by pressing the graphics pen in the LOAD block. For the file name type NET <CR>. As soon as you press the return key, the computer will begin to read a file called NET which was previously SAVEd to your disk. As it is read in, the text screen lets you know which part of the file is being read, and the graphics screen starts to fill up. As soon as the file has been read in completely, you will see the "NO MODE ACTIVE" message on the text screen. That means the computer is ready for another selection.

Use the pen to select the SAVE command. For a filename use your first name, then <CR>. The graphics screen will not change, but the text screen will tell you which part of the file it is currently writing. As soon as the network's data structure is completely written to the file, you will see the "NO MODE ACTIVE" message. You now have two files on your disk containing network data.

Let's next try LOADing the file you just SAVEd. But, what must we do first? Select the CLR command and answer the question with "Y". Notice how the graphics screen is cleared and you are returned to no mode active. Now select the LOAD command and enter your first name as the filename, then <CR>. The network reappears on the grahics screen.

Feel free to experiment some more with these commands. When you want to quit this program, select the EXT command. By answering the questions that follow you will easily and naturally end the session.

LIST

Before we look at the actual QGERT symbols, we'll describe the LIST command. By selecting it you can see a list of available QGERT symbols on the text screen.

Now, let's look at the QGERT symbols.

SOU, REG, STA, QUE, SIN

These are the nodes that you can select. They are respectively, SOUrce, REGular, STAtistics, QUEue and SINk nodes. Upon selection of any of these, the text screen will notify you of the type node you have selected. At this point the computer is waiting for you to depress the graphics pen a second time. If you depress the pen in the graphics area of the tablet, then the node you selected will appear in that spot on the graphics screen. If, however, you depress the pen outside of the graphics area on the tablet, you may lose the present node selection. For example, if you depress the pen in another menu block, you will obtain that command/symbol. As soon as the symbol is drawn on the graphics screen, the text screen will start to ask you various questions. As you answer these questions, some of the answers will appear on the graphics screen in the appropriate spot to further define your node. Other answers will not appear on the graphics screen but will be included in the data structure for later use. When you have answered the last question, you are returned to the familiar no mode active screen.

NOTE: The character file available for writing text on the graphics screen does not include the infinity symbol. Therefore, you will see a "-" on the graphics screen if your answer to a question is infinity.

ACT

The ACTivity symbol is slightly different. After selecting it you are told that you are in the activity mode. You do not have to depress the pen a second time in the graphics area of the tablet as you did with the nodes. You will immediately be asked some questions. As you answer them, you will see your activity drawn between the nodes indicated by your answers. Again, only that information usually seen on a network will be drawn on the graphics screen, but all answers will be placed in the data structure to be used later. After the last question is answered, you are returned to the no mode active screen.

NOTE: Activities on the graphics screen are drawn only as straight lines. So, keep this in mind as you position the start and end nodes of an activity. Also, before attempting to draw an activity, be sure you have drawn BOTH the start and end nodes of the activity.

Exercise #2

In this exercise, you will make up your own network. Don't worry if you don't have a real system in mind to model. All you want to do is to get familiar with placing some symbols on the graphics screen. Start by placing at least two nodes (one at a time of course) on the graphics screen. Answer the questions anyway you want within the indicated limitations. Then, connect these two nodes with an activity. Continue this until the graphics screen begins to get full. At this point you would have to quit. If you want, use the SAVE command to save this network. Then, whether you SAVEd or not, use CLR to clear the graphics screen. Make as many networks as you need to get familiar with the tablet and these symbols.

EDIT

Now that you can create your own networks, you probably want to be able to change or correct a network you have SAVEd to disk. This is done with the EDITor. It allows you to change parameters in a node or activity, or you can delete the symbol entirely. Notice the editor is not used to add a QGERT symbol; this is done by LOADing the network and then adding the wanted symbols to the existing network.

Before you select the EDITor command, make sure that you have LOADed the network that you want to change. If you want to EDIT the network you are currently working on, you must SAVE it before entering the EDITor. You will be guided through various menus to make your corrections/deletions. In the main menu of EDITor, there is an option called UPDATE. After you have made several changes you can update the graphics screen to reflect these changes. However, if you choose this option, the changed network will be SAVEd to the filename from which it was LOADed/SAVEd. Therefore, do an UPDATE only if you don't mind replacing the original network with the changed one. Upon leaving the EDITor, it makes sure you have saved the network to some filename and then returns you to the no mode active screen.

LOCK X, LOCK Y, CLR LOCK

LOCK X and LOCK Y allow you to lock the x or y coordinate for subsequent graphing. For instance, you may have several nodes which you want to line up in the exact same column. This is easily done by locking the x coordinate. No matter where the pen is positioned in the graphics area, the node on the graphics screen will have the x coordinate that was locked. In the same way, a horizontal alignment can be accomplished by LOCK Y. Although these two commands are for cosmetic purposes only, it's a good idea to learn how to use them. They are extremely useful in positioning a new node in place of a deleted one. After you are finished using a LOCK X or LOCK Y, you clear the lock by using CLeaR LOCK.

Exercise #3

This exercise will get you familiar with the EDITor and LOCK's. You need to start with a network. So, either LOAD a previous one or create a new one. Now select EDIT. When you get to the main menu, decide if you want to change a node, acitivity, or parameter set. Indicate your decision by typing in the appropriate letter. You then indicate which symbol # you want to change. The EDITor will then find your particular symbol and display the information for that symbol. Your first option is deletion. Type N for now. Since you indicated you do not want to delete the entire symbol, you must want to change some piece of the symbol. You are given some instructions which explain how you can make these changes. Notice as you make these changes, the information displayed at the top of the screen is immediately updated; the graphics screen is not. When you are through changing this particular symbol, type "Q" to indicate quit. You will be returned to the main EDITor menu. You can quit the EDITor, change another symbol, or UPDATE your graphics screen with the changes you have already made. Before you quit the EDITor, delete one of the nodes (REG, QUE, or STA), and UPDATE the graphics screen. Quit the EDITor and try to replace the empty spot left by the deletion with a new node by first locking x or y and then selecting a node symbol (REG or STA).

Experiment with the EDITor and LOCK commands. The more you do, the better you will feel about this program.

GEN

You have now been introduced to all the commands/symbols that will enable you to draw a network. Once you have completed a network to your satisfaction, you can GENerate the QGERT source code. This is where the real savings (time and frustration) appear in this software package. By selecting the GEN command you enter a new program automatically. It starts by asking you which file you want to load. Respond by typing the name of the network for which you want to GENerate source code. The program loads the named network, and then begins asking you a series of questions. You will recognize these as fields in the General card. After these questions, the program writes the source code to a text file with the same prefix as your network's name and shows you exactly what was written to this file. Finally you are automatically taken back to the graphing program (QGERTNET) where you can draw another network, GENerate more source code, or quit the session.

<u>Exercise #4</u>

For this exercise, practice any of the commands/symbols you want. Above all, try the GENerate command on several networks. You might want to draw a network, SAVE it, GENerate source code, EDIT the network, SAVE it, GENerate source code, etc.

Conclusion

In order to make back-up copies, delete unwanted files, perform other general Apple Pascal disk operations, you w have to know a little about the Apple Pascal Operat System. Since this manual is not designed as a text Apple Pascal, you are referred to the following manuals:

1) Apple Pascal Operating System Manual
2) Apple Pascal Language Reference Manual

Don't get discouraged if you're not familiar with Ap Pascal. As stated in the Introduction of this manual, don't need to know Apple Pascal to draw QGERT networks generate source code with this software package.

You have now been introduced to the full range of comma and symbols of this sosftware package. Remember to pract with your tablet. As you do, you will become more skillf and simulating will become easier.

Bibliography

1. "All About Personal Computers". DataPro. DataPro Research Corporation. Delran, N.J. (April 1981).

2. Apple Pascal Language Reference Manuals. Apple Computer, Inc. Cupertino, California, 1980.

3. Apple Pascal Operating System Reference Manual. Apple Computer, Inc. Cupertino, California, 1980.

4. Bylinsky, Gene. " A New Industrial Revolution Is On The Way," Fortune: 96-104 (Oct 1981).

5. Clark, Thomas, Lt Col. Lectures in SM 6.66, Systems Simulation. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1983.

6. Conway, Richard, David Gries, E.C. Zimmerman. A Primer on Pascal. Cambridge, Mass.: Winthrop Publishers, 1981.

7. "Desk-top Computer Aided System Designs Any Digital System or Component," Military Electronics/Counter-measures: 60+ (Oct 1982).

8. Dorce, Lawrence A.. "Bringing CAD Closer to the Electronic Designer." COMPCONS, IEEE. pp 255-259 (Spring 1982)

9. Ellison,David, Irma Herschdorfer, Jean Tunnicliffe Wilson. "Interactive Simulation on a Micro Computer." Simulation, Vol 38: pp 161-175 (May 1982).

10. Graphics Tablet Operations and Reference Manual. Apple Computer Inc. Cupertino, California, 1979.

11. Keller,James H.. " A System Design Tool For Automatically Generating Flowcharts and Preprocessing Pascal," Wright-Patterson AFB,Ohio: Air Force Institute of Technology; 1979. Master's Thesis.

12. McKenney, J. L. "A Clinical Study of the Use of a Simulation Model." The Journal of Industrial Engineering, Vol XVIII. No. 1, (Jan 1967).

13. McLeod, John. Simulation. McGraw-Hill Book Company. New York, 1968.

14. Pritsker,A. Alan B.. Modeling and Analysis Using Q-Gert Networks(Second Edition). New York: Halsted Press, 1979.

15. Scallon, G.M.,J.A. Grupe. "Functionally-Oriented System Simulation for Computer Aided Design of Software/Hardware Systems." Simuletter Vol 2,3,4: pp 51-65 (Winter,Spring,Summer 1978).

16. Schoderbek, Charles G.,Peter P. Schoderbek, Asterios G. Kefalas. Management Systems Conceptual Considerations (Revised Edition). Dallas,Texas: Business Publications, Inc., 1980.

17. Shannon, Robert E.. Systems Simulation: the Art and Science. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1975.

18. Sippl, Charles J. Micro Computer Handbook. Mason Charter Publishers, Inc. 1979.

19. Sokol, Dan. "Computer-Aided Drafting with Apple Pascal." BYTE. pp 388-429. (July 1981).

20. ______ "Notes on Absolute Location Interfaces to Apple Pascal." BYTE. pp 324-325, (September 1980).

21. Spoonamore, Janet, Kenneth Crawford, Edgar Neely, Jr. "Computer Aided Engineering and Architectural Design," The Military Engineer,74: (April 1982).

22. Sutherland, Ivan E. "Computer Displays." Scientific American. (June 1970).

VITA

Captain Gary M. Anderson was born on 24 December, 1952 in Mt. Pleasant, Iowa. He graduated from high school in Wichita Falls, Texas in 1971. He attended Parkland Junior College and the University of Illinois, studying Electrical Engineering. In 1977 he received a Bachelor of Science degree in Accounting. On May 18, 1978 he was commissioned into the USAF through OTS. He was stationed at Hanscom AFB as a Budget Officer and Program Analyst for ESD. He then spent three years at Eielson AFB as the MWR Financial Management Officer. In June 1982 he entered the School of Engineering, Air Force Institute of Technology.

Permanent address: 2410 Carrelton
Champaign, IL 61820

Captain Chris R. Commeford was born on 3 January 1952 in Honolulu, Hawaii. He graduated from high school in Kailua, Hawaii in 1970 and attended the United States Air Force Academy. After graduating in June 1975, receiving a Bachelor of Science degree in Engineering Management, he entered pilot training at Columbus AFB, Mississippi. Receiving his wings in September, 1976, he became a B-52 pilot serving tours at Minot AFB, North Dakota and Carswell AFB, Texas prior to entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 45-120A Mololani Place
Kaneohe, Hawaii 96744