DEVELOPING A SIMULATION LANGUAGE FOR FLEXIBLE MANUFACTURING SYSTEMS

By

William Henry Remy, III

Bachelor or Science

Oklahoma State University

Stillwater, Oklahoma

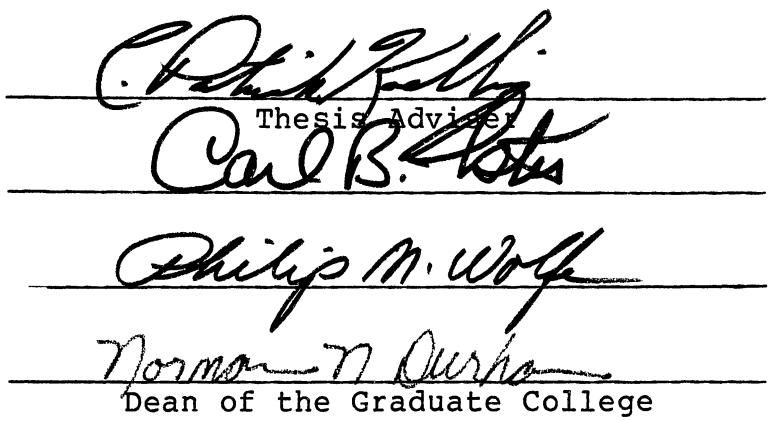
1982

Submitted to the Faculty or the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements or
the Degree of
MASTER OF SCIENCE
May, 1984

DEVELOPING A SIMULATION LANGUAGE FOR

FLEXIBLE MANUFACTURING SYSTEMS

Thesis Approved:



Thesis Adviser

Dean of the Graduate College

PREFACE

This research was focused on developing an interactive simulation package for analyzing flexible manufacturing systems. The research consisted of three phases. The first phase consisted of identifying and defining the current status of flexible manufacturing systems as they exist in industry. In the second phase, the characteristics and capabilities of simulation languages were examined. The third phase then integrated the information obtained from phases one and two to provide the actual software. The software consists of over 100 subroutines for model description, model editing, model simulation, and system reporting.

The results of the research were quite promising in the sense that a very useful tool was developed. Also, as is the case with most new software, several areas for addition and improvement were identified. Two of these areas were possibly adding graphics capabilities and adding a statistical analysis module.

I wish to express my sincere gratitude to Dr. C. Patrick Koelling, my major adviser, for providing constant support, guidance, and invaluable assistance which helped to make this research effort successful.

I am also indebted to the other committee members, Dr.

Carl B. Estes and Dr. Philip Wolfe, for their guidance during the course of this research.

A special thanks is due to the School of Industrial Engineering and Management for allowing me to pursue an advanced degree and for the financial support that was provided during my stay at Oklahoma State University.

The information on flexible manufacturing provided by Dr. Neal McCollom proved to be invaluable in the first phase of my research. Also, I would like to thank my colleagues Hans Demmel, John Lewis, and Clay Thompson for their patience and insight.

I would like to thank Dr. Joe Mize for providing me with the simulation knowledge and the motivation for attempting an endeavor such as this.

My wife, Janis, deserves my deepest recognition for her unending support and understanding during the last two years.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

LIST OF TERMS

AGV - Automatic Guided Vehicle

AGVS - Automatic Guided Vehicle System

AS/RS - Automatic Storage/Retrieval System

CNC - Computer numerical control

DNC - Direct numerical control

FMS - Flexible Manufacturing System

FMS - Flexible Manufacturing Simulator

$\log_2$ - Base 2 logarithm

NC - Numerical Control

$N^*(0,1)$ - Normally distributed with mean 0 and standard deviation of 1

Queue - Waiting line

U[0,1] - Uniformly distributed on the interval 0 to 1

CHAPTER I

THE RESEARCH PROBLEM

Purpose

In recent years the economic structure of American industry has deteriorated. The major reason for this decline is the inability of American industry to keep abreast of new developments in technology. An area of technology that is currently receiving a great deal of attention is the area of information processing. Specifically, engineers and managers are interested in gathering and analyzing information about the performance of production operations. A popular method for analyzing the inter-relationships in a manufacturing environment is computer simulation.

The area of computer simulation is relatively new and still a bit primitive in some aspects. One area of simulation that is lacking is the area of analyzing manufacturing systems. This research is focused on three objectives: first, to identify the reason why simulation methodology is out of phase with manufacturing technology; secondly, to identify the strengths and weaknesses of computer simulation in the area of manufacturing systems; and finally, the main objective is to develop a simulation

language that addresses current strengths and weaknesses in simulation methodology to provide manufacturing analysts with a state of the art tool.

Introduction

Information in business exists almost without bound. Every day more and more information is generated concerning all aspects of a business. Consequently, everyone from first line supervisors to chief operating officers is becoming buried in the flood of information. To combat this situation new tools for gathering and analyzing information are needed. These new tools should utilize the speed and accuracy of the digital computer.

In the United States, firms engaged in the manufacturing of consumer goods, such as automobiles, are very concerned with the performance of their daily operations. To assess the performance of manufacturing operations, whether existing or proposed, a firm must employ certain information gathering and analysis techniques. A technique that has recently become popular for gathering and analyzing information about manufacturing systems is computer simulation; specifically, discrete event simulation. Discrete event simulation is used since a large portion of the items produced in the United States are described in discrete units of measure. Also, the activities undertaken by most manufacturing firms are measurable in discrete terms.

Current State of Simulation

Discrete event simulation has been present in the academic domain for approximately 30 years. However, it has not been until the past five to ten years that it has gained recognition in the industrial environment. Simulation has proven to be a useful tool in analyzing the structure and performance of manufacturing environments, both proposed and existing conditions. However, the manufacturing environment is undergoing rapid change due to the increased competition from domestic sources and the international markets. Consequently, many of the early simulation languages have become somewhat obsolete. In addition to not maintaining pace with the changes in manufacturing, simulation languages have not maintained communication with the computing world.

Many simulation languages on the market today were not designed to be both effective and efficient. The effectiveness of a language is how well it can be used as a decision-making tool. A good simulation language should provide the engineer or production analyst with a tool for gathering and analyzing information about a specific system. The efficiency of a language is two dimensional. In the first respect, an efficient language should allow the user to achieve a quick and accurate estimate of system performance. Secondly, an efficient language should minimize the amount of computer resources necessary to acquire the desired information about the system.

The area of computer simulation needs to monitor the

needs of manufacturers and advances in computer science. Many of the simulation languages available today were designed by the so-called experts. The result of the designs are such that the users of the languages must possess advanced skills in simulation. Another aspect of many languages is that they employ only simple data structures and data manipulation techniques. Consequently, these simulation languages use a lot of computer resources unnecessarily. Simulation languages seem to have fallen into a pattern of design that has led them in a direction away from the needs of manufacturers and developments in computer science.

The design of future simulation languages should provide the practitioner with an easy to use tool for analyzing the structure and performance of manufacturing systems. A simulation language should not require a user to possess advanced skills in simulation methodology. Also, any new languages should attempt to employ advanced techniques in computer science. Obviously, it would be impossible to use the most recent developments since new developments are made with such a high frequency.

Current State of Manufacturing

Many manufacturers in the United States are taking great steps toward becoming competitive again. The advances being made in manufacturing are occurring rather rapidly. However, the general direction of manufacturing is quite

predictable. It appears the era of giant factories producing large quantities of similar items has passed. The future of manufacturing lies in producing smaller quantities of goods with a large variety. Consequently, firms are employing a method of production called flexible manufacturing.

Flexible manufacturing systems (FMS) are a network of machines and transfer mechanisms. A flexible manufacturing system can be as simple as a robot loading and unloading two numerical control lathes. On the other hand, a flexible manufacturing system could be as complex as fifty machine cells, similar to the aforementioned example, all linked together by a system of automatic guided vehicles. Koren (18) defines a flexible manufacturing system as a network of machine cells. He also goes on to state that a machine cell is a combination of computer numerical control (CNC) machines and robots. In practice, machine cells would probably contain regular numerical control machines, conventional machine tools, and transfer systems other than robots. In a more detailed description of flexible manufacturing systems, Groover (12) identifies three basic elements: the machine tools, the material handling system, and the computer control system. He also identifies two types of flexible manufacturing systems: dedicated and random.

The design and structure of flexible manufacturing systems are such that they can be viewed as a network of

elements. These elements can be an entire machining cell or simply a direct numerical control (DNC) lathe. The elements within a flexible manufacturing system either contain various material handling devices, or are joined by them, or both. The FMS environment is a natural application for simulation modeling due to the relationships between the machines, transports, and entities that pass through the FMS.

Summary

The previous discussion of flexible manufacturing systems shows that simulation could indeed be a valuable tool for analyzing such systems. However, languages currently available in the industrial domain do not meet the criteria established previously. The lack of a good simulation language for analyzing flexible manufacturing will provide the basis for this research project. Specifically, this research project will focus on developing a good simulation language for the analysis of flexible manufacturing systems. The language should include features mentioned previously along with those presented in the next section.

CHAPTER II

REVIEW OF THE LITERATURE

Introduction

This chapter is divided into two main sections. The first section deals with literature specific to simulation methodology. It will be focused on general and specific characteristics of simulation languages. The second section will present the literature associated with flexible manufacturing systems. This section will provide definitions and specific characteristics intrinsic to flexible manufacturing systems. It will also provide a basis for the scope of this research.

The Simulation Literature

The area of computer simulation is a relatively new and rapidly growing field of study. Computer simulation began in its earliest stages on analog computers. However, with the development of digital computers simulation entered a new era. Another advancement in the computer industry that aided in the early growth of simulation was the development of high level programming languages such as FORTRAN and PL/I. According to Shannon (25), early simulation models were very machine dependent and written in low level machine

languages.

Once modelers were adept at using the general purpose programming languages or base languages, such as FORTRAN, they began to notice some similarities between models. These similarities formed the early basis for constructing specialized languages for building simulation models. Even though simulation methodology had made another improvement with the development of specialized languages, there are still some disadvantages. Shannon (25) points out that the early simulation languages were developed by individual companies and were not readily available to the commercial market.

The basic function of a simulation language, as identified by Gordon (10), is to provide a data structure to form the problem into a model. A simulation language should provide the modeler with the vocabulary and syntax for constructing a model. Shannon (25) identifies the following seven features of simulation languages as compared to general purpose or base languages:

1. Reducing the programming task;
2. Providing conceptual guidance;
3. Aiding in defining the classes of entities in the system;
4. Providing a means of differentiating entities by characteristics;
5. Providing the flexibility for change;
6. Describing the relationship of entities to one another and to their environment;

7. Adjusting the number of entities according to changes in the status of the system.

Shannon also points out that a specialized simulation language can provide a good method for communicating and documenting a model. This can be a critical point during model validation, especially when presenting the model to non-technical personnel.

In the previous discussion, several common attributes of simulation languages were presented. However, the common characteristics mentioned are quite general and not exhaustive. Emshoff and Sisson (5) list seven specific functions of a simulation language (Table I). This list of functions is by no means exhaustive; however, it does provide a starting point for comparing the usefulness of different simulation languages.

TABLE I

SPECIFIC FUNCTIONS OF SIMULATION LANGUAGES

Create random numbers on the interval Uniform (0,1).
Create random varieties.
Provide a time keeping mechanism.
Collect statistics throughout the simulation.
Perform statistical analysis on the output.
Provide output in predetermined formats.
Detect and report logic and syntax errors.

Specific Design Criteria for Simulation Languages

The comparison of different simulation languages is multidimensional. In order to effectively present the different design criteria and identify the important criteria, the analysis of design will begin at a macroscopic level. The analysis will then progress to a very detailed level, addressing such items as sorting and searching event lists.

Differentiating the "World Views". A simulation language is only useful when the problem cannot be easily formulated using a base language, such as FORTRAN. Once the decision has been made to use a simulation language instead of a base language, there are several design criteria that can be used to select the appropriate language. The first criteria in selecting a simulation language is the type of "world view" that is employed by the language. The term "world view" refers to whether the system will be modeled using a discrete, continuous, or combined discrete-continuous framework. The areas of continuous and combined discrete-continuous simulation modeling are very interesting; however, this research is focused on the area of discrete simulation modeling.

In discrete simulation modeling, there are three additional views: event orientation, activity scanning, and process orientation. According to Pritsker (23), in the

event-oriented world view a system is modeled by defining the changes in the system that occur at event times. The modeler is then faced with the tasks of determining the set of events that can change the system, and developing the associated logic. This logic is usually in the form of FORTRAN subroutines.

The activity scanning orientation determines which activities in the system entities can engage in. The modeler must then prescribe the conditions which will cause an activity to begin or end. Pritsker (23) states that this view is somewhat inefficient since each activity must be scanned at every time advance. He also states that this method is not widely used; however, many languages employ some feature for determining activity times based on system status.

In the process view of discrete simulation, systems are modeled using a set of predefined elements. These elements consist of such items as queues and servers. The system can then be described by a set of formatted statements corresponding to one of the predefined elements. The coded statements are then executed automatically by the simulation language. The process orientation will often employ both the event and activity scanning orientations.

Comparison of World Views

The three different world views within discrete simulation have appropriate applications within different

modeling schemes. The event orientation is by far the most flexible of the three views. However, it also requires the highest level of simulation skills. The activity scanning orientation is appropriate when activity times are indeterminant. The use of activity scanning is limited by the inefficiencies of computer resources. Also, activity scanning is limited by the small range of physical applications. The process orientation of discrete simulation does not provide the flexibility of event orientation; however, it does have more practical applications than activity scanning. Process orientation also requires less programming effort than the event orientation.

Additional Design Criteria

The selection of a discrete event simulation language is an important decision to a company. To make an intelligent decision the firm must consider the inherent design characteristics of the available languages. The major categories of design considerations mentioned by Pritsker (23) are presented in Table II.

The items in Table II provide a comprehensive list for evaluating a simulation language. However, within each category there are several tradeoffs to be considered.

Amount of Training Required. The amount of training required to use a simulation language is a function of the world views employed. If a language were only to employ the

TABLE II

COMPARATIVE DESIGN CRITERIA FOR SIMULATION LANGUAGES

Training Required	Ease of learning the language Ease of problem conceptualization
Coding Considerations	Ease of coding, including random sampling and numerical integration The degree to which code is self documenting
Portability	Language availability on new or existing computers
Flexibility	The degree to which a language supports different modeling concepts
Processing Considerations	Built-in statistics gathering List processing capabilities Ability to allocate core memory Ease of producing standard reports Ease of producing user generated reports
Debugging and Reliability	Ease of debugging Reliability of compilers and support systems, documentation
Run-time Considerations	Compilation speed Execution speed

process orientation, the ease of learning would be straightforward. Also, the ease of conceptualizing systems would be relatively straightforward. On the other hand, a language that employs the event orientation would require more training and would not provide straightforward model conceptualization. However, the capabilities associated with event orientation are numerous. The amount of training required for the activity scanning orientation would be similar to that for the event orientation.

Coding Considerations. The ease of coding is again a function of the world views employed by a language. The process orientation would be the best in both categories under coding considerations. The reason for this is that the process orientation relies strictly on a set of predefined statements, whereas the activity scanning and event orientation are dependent on the base language being used.

Language Portability. The computer explosion of the last three years has diminished the importance of this design criteria. Any simulation language with a base language of either COBOL, PL/I, or FORTRAN would be highly transportable.

Language Flexibility. The degree to which a language supports different modeling concepts is dependent on the world view(s) employed and the purpose of the language. Some languages on the market today employ more than one

world view whereas others do not. Also, certain languages on the market are designed to simulate special type systems. The tradeoff in flexibility is realized by determining the types of models that are needed and determining the set of capabilities needed to construct the models. For example, a manufacturer of engines that employs both assembly line and job shop production methods might require a general purpose simulation language with more than one world view. Two of these languages are SLAM (23) and SIMAN (22). In another instance a firm may need a specialized language with one world view such as GPSS (23).

Processing Considerations. The area of language processing considerations is divided into five categories. These five categories provide the majority of important design criteria. The first category is concerned with how easily a language allows statistics collection within the model. The statistics gathering function can be further divided into those statistics which are collected automatically and those which are requested by the modeler. Any well designed simulation language should provide both types of statistics generation.

The list processing capabilities of a language refers to the queueing disciplines allowed and the internal data structures employed to implement the different event manipulations. A well designed language should allow numerous queueing priorities to facilitate the modeling of a

variety of situations. From the internal list processing viewpoint, the algorithms used should be both effective and efficient. An effective list processing algorithm is one which accomplishes the task with certain economical coding considerations. In other words, the algorithm should be complex enough to perform the task accurately without requiring a lot of coding time. The efficiency of an algorithm is measured by both the execution speed and the memory requirements.

The ability of a simulation to control the amount of core memory allocated during a specific model execution is critical. The cost of computer resources is very high and can often be a limiting factor in the development of simulation models. Consequently, core memory must be allocated and de-allocated as needed.

The last two categories are concerned with report generation. The availability of standard and user written reports is an important element in a well designed simulation language. A new aspect of computing is the area of interactive report generation. This idea will become a standard in future simulation languages.

Debugging and Reliability. The debugging capabilities of a language are very important to the end user. Also, the debugging capabilities of a language are closely related to the ease of learning the language. A good simulation language should differentiate errors and provide accurate and detailed messages. The messages should also indicate

possible causes and the associated corrections for the error types.

The reliability of a language is related to the number of cases which the compiler and language can handle. Also included in this is the issue of language documentation. How well are the features of the language documented and how accurate is the documentation?

Run Time Considerations. The run time considerations associated with a language are divided into two categories: compilation speed and execution speed. The well designed compiler for a simulation language should convert the model statements into the executable form rapidly and accurately. Also, the compilation and execution speeds will be a function of model size and complexity. However, a good language should provide an economical use of computer resources.

The Flexible Manufacturing Literature

Defining a Flexible Manufacturing System

The manufacturing environment in the United States is in the process of switching to a method of production known as flexible manufacturing. In an article written by Hegland (14), it is stated that by the year 1990, more than 50 percent of the machine tools produced will find no independent use, but will be part of flexible manufacturing systems. He also states that by 1985 on-line optimization

of entire plants, controlled by a central computer, will be a fact of life. The definition of a flexible manufacturing system as given by Kearney and Trecker (16) is as follows:

> An FMS is a group of NC machine tools that can randomly process a group of parts having different sequences and process cycles using automated material handling and central computer control to dynamically balance resource utilization so that the system can adapt automatically to changes in part production mixes and levels of output (p. 53).

The definition given by Kearney and Trecker is a general one and needs some clarification. In an article appearing in Manufacturing Engineering (6), two specific definitions of flexible manufacturing systems are presented. The first definition given is for the classical FMS and is as follows:

> This form of FMS . . . is an automated production system for the manufacture of midvolume, midvariety workpieces, consisting of a number of machine tools (usually machining centers, head changers, head indexers, etc.) tied together by an automated workpiece handling system, all controlled by a central computer. The central computer down loads NC programs to individual machine tools in the system, controls workpiece flow, and generates performance reports (p. 49).

The second definition given describes a new type of flexible manufacturing as a stand-alone unit. The system usually is comprised of a single machining center or CNC lathe equipped with carousels or rotary tables for part storage. This definition of flexible manufacturing appears only to be a subset of the previous one. Therefore, it will not be addressed further. The definition given by Manufacturing Engineering is more specific than the

definition given by Kearney and Trecker. However, there are a couple of terms that need clarification.

The definition of midvolume was not given in the Manufacturing Engineering article. However, Hegland (13) and Klahorst (16) state that the production levels may range from 5000 to 100,000 parts annually. In reference to the variety of workpieces processed, the numbers range from one to fifty. The case where only one type part is processed by the FMS is identified as a dedicated FMS by Groover (12). In the case when the number of different workpieces exceeds one, this is classified as a random FMS by Groover (12).

This section has presented a definition of flexible manufacturing. In the next sections the structure of a flexible manufacturing system will be discussed, along with the information and material flows within an FMS.

The FMS Structure

In this section major components of a flexible manufacturing system will be identified. Also, a discussion of the specific characteristics of each component and the interactions between components will be presented.

The major components of a flexible manufacturing system, as identified by Groover (12), are the machine tools, the material handling system, and the computer control system. As previously mentioned, an FMS may be either dedicated or random. In the case of a dedicated FMS the machine tools employed will be more specialized in

nature. The reason for this is that a known product with known processing requirements will be manufactured for some time frame. In contrast, the machines employed in a random FMS will be more general, thereby allowing greater flexibility in the variety of workpieces that can be processed. The machines employed in either type of FMS will be similar in the sense that they will all be computer controlled. The use of manual controlled machines in an FMS is very limited.

Processing Considerations

The number of machines present in an FMS usually does not have an upper limit. However, the average number of machines used in the flexible manufacturing systems currently in use is around fifteen. Klahorst (16) presents three rules concerning the number and selection of machine tools for use in an FMS environment. First, he states that a true FMS requires a minimum of four machines. Any less than four machines would be considered a stand-alone or phase I manufacturing cell. Secondly, Klahorst (16) states that special process machinery should not be considered unless it can do the work of two machining centers in the system. His reasoning behind this rule is that the objective is to be as flexible as possible for as long as possible. Also, the capital should only be dedicated when the economics of scale are present. The third and final rule presented is that any positional tolerance less than

0.001 inches should not be performed using the FMS. Also, any tolerance below 0.002 inches will require special processing considerations in order to maintain size. These tolerances are a result of the large number of variables with which the FMS must contend.

Workpiece Movement Considerations

A flexible manufacturing system usually contains two types of material handling systems. The first type is for moving parts between work centers. The second type, which is really a subset of the first type, are mechanisms used for moving parts within work centers. These material handling devices, referred to as shuttles by ElMaraghy (4), provide buffer stocks of parts between the work centers and the main material handling system. Shuttles may consist of such devices as rotary tables and robots. In contrast, the types of material movement devices found in the primary material handling systems are conveyors, stacker cranes, tow-carts, and guided vehicles. The material handling system employed within a flexible manufacturing system must meet certain operating characteristics as defined by Groover (12). The first attribute Groover mentions is that the material handling system must provide for random, independent movement of palletized workparts between work stations. The term "random" signifies that parts must be able to flow from one station to any other station. It is assumed by Groover that all parts will be attached to some

type of fixture during movement. The term "independent movement" means that a palletized load must be able to move independent of other pallet loads. The second attribute of the material handling system is that is should provide temporary storage of work parts. This attribute is similar to the idea of a shuttle presented by ElMaraghy (4). The third attribute is that the material handling system should provide easy access for loading and unloading parts. Again, this idea relates to the shuttle concept. The use of manual load/unload stations is included in this third attribute. The last two relevant characteristics presented by Groover are that (1) the material handling system must be compatible with computer control, and (2) the system must be expandable on a modular basis. The compatibility of the material handling system with computer control will be addressed in the subsequent section.

The Computer Control System

The computer control system is probably the most critical element in the structure of an FMS. The computer control system's main function is that of system coordinator. Associated with the function of system coordinator Hegland (14) identifies four tasks performed by the controlling computer.

The first function Hegland (14) mentions is that of data distributor. This task involves the bi-directional transfer of data between the machines and the central

computer. He mentions that the data is usually in the form of NC part programs or operator messages. Closely related to the data distributor is the task of traffic coordination. According to Hegland, the task of traffic coordination involves controlling the movement of workpieces between work centers. The remaining two functions of the central computer are those of the tool manager and of work preparation. The tool manager stores and updates data files regarding tool life and compensation characteristics. The work preparation task is concerned with the sequence of events necessary to load a pallet(s) into the system.

The areas of data distribution and traffic coordination can be further sub-divided by different functions according to Groover (12). The data distribution has two functions according to Groover, which are the same two as identified by Hegland (14). However, in the area of traffic coordination Groover mentions three functions which Hegland does not. The first area is that of traffic control. This function is concerned with monitoring and controlling the flow of individual workpieces in the primary material handling system. The control of the material handling system is accomplished by dividing the system into "zones". A zone is a section of the primary material handling system that is defined by two work centers as the starting and ending points. The control is established by allowing only a certain capacity of pallets into a zone at any time. The central computer can activate switches and shuttles to

control the flow into and out of zones in the primary system. The second function of traffic coordination mentioned by Groover (12) is that of shuttle control. Shuttle control involves controlling the material handling devices associated with a specific machine. Finally, the third function is that of work movement monitoring or pallet control. This involves keeping track of the location of each and every pallet. This function is necessary since each pallet is fixtured to handle only certain varieties of parts.

Data Control and System Reporting

In this section the relevant data files used by the central computer in an FMS are discussed. These files are the routing file and the pallet reference file. Also included in this section is a discussion of the reports and information provided by the central computer with respect to system performance. All of the information appearing in this section was drawn from Groover (12), Mize (21), and McCollom (20).

The routing file for a given FMS contains the primary work stations where a part is to be processed. This file will also contain alternatives to the primary routing. The computer can then select the proper sequence for a part to follow given the status of the system. The decision to go to a particular machine other than the primary machine is determined using a specific machine dispatching rule. An

example might be as follows: Select the machine with the largest amount of idle time. These dispatching rules are based on machine availability, queue sizes, and system congestion. At each specific machine part selection rules are employed. An example of these rules are shortest-processing-time-first and first-in-first-out. The pallet reference file maintains the data for all pallets used in the system. It maintains the number of each pallet type available for a given part type.

In the area of system reporting there are three main reports that are of interest: the utilization report, the production summary report, and the status report. The utilization report provides information on the usage of individual machining centers and of the entire FMS. Production summary reports provide information concerning output quantities by individual part type and for the system as a whole. The system status report provides instantaneous values of system utilization and system parameters.

Summary

In this chapter a review of the literature relevant to both computer simulation and flexible manufacturing is presented. The simulation literature provides a background for developing basic design criteria for simulation languages. Also, the simulation literature provides a method for comparing the different features associated with a language. The flexible manufacturing literature provides

insight as to the structure of a flexible manufacturing system. The literature identifies four main components of a flexible manufacturing system: the work centers, the material handling system, the computer control system, and the workpieces. The literature also provides a knowledge base for understanding the logical structure of an FMS. The logical structure involves the transfer of information and data within the system.

The information derived from the literature review will be collated and integrated to develop a new simulation language for analyzing flexible manufacturing systems. The design criteria and language boundaries will be presented in the next chapter.

CHAPTER III

DEFINING THE RESEARCH SCOPE

Introduction

In this chapter the proposed simulation language will be described. To determine the scope of this new language a definition of flexible manufacturing systems specific to this research will be presented. This definition will provide the parameters for the new language and the boundaries for this research. After the new language has been presented it will be compared to several existing languages in order to illustrate the contribution of this research to the field of simulation.

Flexible Manufacturing Redefined

The definition of a flexible manufacturing system for the purposes of this research will be derived from the definition previously given by Manufacturing Engineering (6). However, to make their definition feasible as a basis for designing a new simulation language, some quantitative measures must be included. The first detail that will be included in the definition will be setting an upper limit on the number of different workpieces that can be processed by a flexible manufacturing system. The upper limit for this

research will be set at 20 different workpieces. This value will provide enough flexibility for practical purposes without making the initial design and development overly complex. The number of machine tools will have a minimum of one and a maximum of 25. The lower limit violates the rule given by ElMaraghy (4); however, if the limit were higher it would restrict the language unnecessarily. The maximum value for the number of machine tools should provide enough flexibility for practical applications. The material handling system definition will be divided into two categories: the primary system and the secondary system. The primary material handling system will be used to move pallets of workpieces between stations. The primary system will be divided into zones as defined by Groover (12). Each zone will be a segment of the primary system with two machining centers as its starting and ending points. There will be only one primary material handling system per FMS. The second category of the material handling system will be the shuttles. The shuttles will act as the load/unload mechanisms for a machine center. They will also provide buffer points for workpieces at any given machine center. The number of shuttles is limited to 75. This number allows the modeler to assign two shuttles per machine center with some extra for miscellaneous tasks. The shuttles will have one or more queues associated with them. Within each queue there will be several queueing disciplines available. Some

of the possible queueing disciplines are presented in Table III.

TABLE III

POSSIBLE QUEUEING DISCIPLINES

Rule	Explanation
FIFO	First in => First out
SPT	Shortest processing time first
LIFO	Last in => First out
LNRO	Least number of remaining operations
SPT/FIFO (a)	Shortest processing time first when the queue size is less than "a"; otherwise use First in => First out

The preceding discussion provides the necessary information to develop a definition of flexible manufacturing specific to this research. The FMS definition for this research is as follows:

> A flexible manufacturing system is an automated production system designed to manufacture from one to 20 different workpieces using from one to 25 NC machining centers or manual work stations. The workpieces will be transported between work centers by a single primary material handling system (usually conveyors or carts) and transported in the work centers by shuttles. Each work center in the FMS will have two shuttles, one for input and one for

output, each having one or more queues for workpieces. The primary material handling system will be divided into zones between each work center with a capacity on the number of pallets in a zone specified. The entire FMS will be controlled by a central computer which sequences the workpieces according to static dispatching and queueing rules. Also, the computer will collect and interpret data on system performance.

A New Simulation Language: **FMS**

The name chosen for the proposed language is Flexible Manufacturing Simulator, or **FMS.** The new language will be developed on the Digital Equipment Corporations VAX-11/780 series computer. The reason for selecting this machine over the IBM 3081D is that the VAX is an interactive computer and the IBM is not. Also, the IBM will not allow a program to create permanent files maintained on peripheral storage during execution. It was decided to use PL/I subset G as the base language for **FMS.** The reasoning behind this was that the use of structures in PL/I greatly reduces the programming task while increasing the flexibility. The world view selected for designing **FMS** was that of a network or process view. This approach was intuitively suggested by the physical and logical relationships that exist in the flexible manufacturing system structure. The level of detail employed in creating models with **FMS** will be similar to that of existing languages. Machine centers will be described by the number of parts that can be processed in parallel at a given time. The duration of processing times may be either constants or random variables drawn from known

distributions. Likewise, the travel times between stations and the load/unload times may be either constants or random variables. The new language will not address the detailed differences between the movement capabilities of a five-axis robot and a six-axis robot. These two entities would be shuttles, each operating with known transfer times that may be constant or random variables.

Major Language Features

The Flexible Manufacturing Simulator will offer three main features that facilitate cost effective model development and system analysis. The first and most important feature of **FMS** is that it is completely interactive. There are clear and accurate menus with easy to understand instructions. Users would not have to be extremely adept at using the computer in order to develop a model. The syntax and validity of parameters are checked immediately upon input. Consequently, the modeler does not have to try and execute the model to find out that a keyword was misspelled. In addition to all model input being entered interactively, the input parameters, model statements, and output reports generated by **FMS** are stored in user named files. The second attractive feature of **FMS** is the ability to generate flexible manufacturing system oriented reports. **FMS** will generate the following four types of reports at the request of the modeler:

1. Utilization Report

2. Production Summary Report
3. Queue Summary Report
4. Breakdown Summary Report

The Utilization Report will provide a breakdown by machine center the time spent on each part type and other items such as idle time; it will also provide information on the overall system utilization. The Production Summary Report will provide information on production rates by individual part type and for the entire system. The Queue Summary Report provides information on the behavior of input and output queues associated with the machining centers. The Breakdown Report summarizes information associated with any service interruptions at the machine centers.

The third feature of **FMS** is the ability to construct models in a modular fashion. This feature is possible since each component of a model, machine centers, shuttles, and workpieces, are all input and maintained independently within **FMS**. As an example, after testing a certain model, the modeler decides to change a dispatching rule for some workpiece. The modeler must simply enter the **FMS** edit mode and change one parameter for one part to evaluate the new model.

Machine Center Description

The Flexible Manufacturing Simulator will support models developed using from one to 25 machine centers. Within **FMS** no differentiation is made between NC machines

and manual machines. The reason for this is that the actual operation is not modeled but rather the delay experienced by a workpart when it is processed at a work center.

In **FMS** there will be three distinct types of work centers: regular work centers, load centers, and unload centers. Regular work centers, abbreviated WC in **FMS**, are where all part processing will occur. A load station (L) designates the work center where parts will enter the system. Conversely, unload centers (UL) are where parts exit the system. In any given model there must be only one load center and one unload center.

Shuttle Description

The purpose of shuttles is to provide secondary material handling either as input/output for a machine center or as input/output to another material handling device. Each shuttle in a model will have associated with it one or more queues. Each queue will in turn have its own queueing discipline. The modeler must specify the work center with which a shuttle is associated. Also, the size of the queue, the queueing discipline, and whether the shuttle is for input or output must be specified. The default value for queue size is infinite with the default queueing discipline being FIFO. However, if the shuttle is not specified as input or output an error will occur.

Workpiece Description

Each part or workpiece that is to be processed by the Flexible Manufacturing Simulator will have a certain machine routing. In **FMS** routings will be determined by operation sequence and dispatching rules. The modeler will specify the order in which individual operations are to be performed. This sequence will then be used as the routing sequence. As previously mentioned, each operation may have one or more machines where the operation may be performed. Using the operation sequence and the set of feasible machines a routing will be determined using the dispatching rules. The possible dispatching rules are presented in Table IV. The processing times at a machine center for a given operation may be either constants or a sample from a random variable distribution. It is assumed that the processing times at stations within a feasible operation set will be equivalent.

Material Handling System Description

The material handling system is dependent upon the placement of work centers and shuttles. Each item that is to be joined by the material handling system can be described by specifying the starting point and then providing the distances between each point in the system. The segment of the material handling system that lies between two points will be referred to as a zone. A zone can provide material flow in one direction or two

directions. The rate of flow for each zone is specified by the modeler. The second item within the primary material handling system which affects its performance is the pallet specifications. The modeler will specify the number of pallets which are available.

TABLE IV

POSSIBLE DISPATCHING RULES

Rule	Definition
SNIQ	Select the machine with the smallest number of parts in the queue(s).
SMHIT	Select the machine with the largest amount of idle time.
Random	Select a machine at random.
LAPT	Select the machine that has the least amount of processing time in the queue(s).

Breakdown Descriptions

Any manufacturing system which employs machine tools as a means of production will experience interruptions in material flow due to machine failures. In **FMS**, machine failures will be handled by specifying three items: the machine name, the breakdown frequency, and the downtime

duration. Also, if desired, the modeler may enter up to a five line description of the breakdown scenario.

Miscellaneous Model Input

In all simulation models there is a certain amount of information associated with the model that cannot be assigned to any one module. This information consists of items such as the modeler's name, the project name, and the current date. Also included in this section of parameter input will be the starting and stopping times for the TRACE option and the model execution segment. The selection of summary reports and the time frame for collecting statistics may also be specified in this input segment.

Comparing **FMS** with Existing Languages

The industrial environment is abundant with simulation languages. In this section **FMS** will be compared to five of the more popular languages currently being used in industry. The five languages that are to be used in the comparison are SLAM (23), SIMAN (22), GPSS (23), SPEED (15), and GEMS (9). The characteristics used to make the comparison will be as follows:

1. Mode of operation (interactive or batch).
2. Level of user computer skills required.
3. Types of systems designed for.
4. Ability to model large scale systems.

In order to facilitate the comparison a language

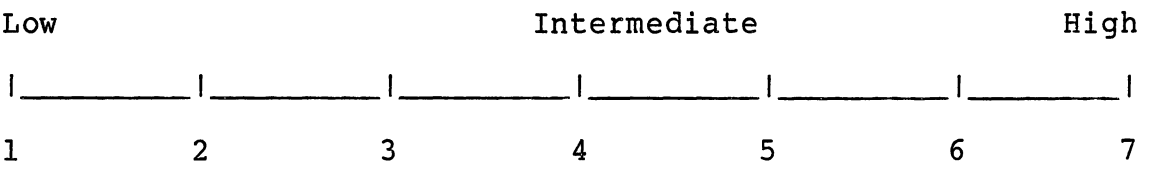
characteristic matrix was developed. This matrix is presented in Table V. The comparison of computer skills necessary to model in a particular language was based on two criteria. The first criteria dealt with the language requiring the user to be knowledgeable in a programming language such as FORTRAN. The second criteria compared the relative amount of training a user would require in order to use a certain simulation language. The scale used to determine the ranking is presented in Figure 1.

TABLE V

LANGUAGE CHARACTERISTIC MATRIX

Simulation Language	Mode of Operation	Level of Skill Required	Types of Systems	Model Large Systems
SIMAN	Batch	4 to 6	GMS	Yes
SLAM	Batch	4 to 7	GPS	Yes
GPSS	Batch	3 to 5	QS	No
GEMS	Batch	4 to 6	GMS	Yes
SPEED	Batch	3 to 6	GMS	Yes
FMS	Interactive	2 to 4	**FMS**	No

GMS ==> Generalized Manufacturing Systems
GPS ==> General Purpose Systems
QS ==> Queueing Systems
FMS ==> Flexible Manufacturing Systems



Low level ==> No prior training in a programming language. Simulation language could be self taught.

Intermediate ==> Some training in a programming language required. Some formal training required to learn simulation language.

High level ==> Must be proficient in a programming language. Training courses required in order to learn the simulation language.

Figure 1. Ranking Criteria for Level of Computer Skills

The new language, **FMS**, fills an important gap in available features among the more popular simulation languages. Although **FMS** is not capable of modeling large scale systems, it does provide the practicing engineer with a good tool for analyzing flexible manufacturing systems.

Summary

This chapter has described the specific characteristics associated with a new simulation language named Flexible Manufacturing Simulator or **FMS**. The types of flexible manufacturing systems which **FMS** will address was derived from the definitions given by Groover (12) and Manufacturing Engineering (6). These definitions provided the basis for defining the language characteristics of **FMS**.

The specific characteristics developed are aimed at the

number of different workpieces that can be processed, the number of work centers, the material handling system, and the computer control system. In **FMS** up to 20 different workpieces can be processed through as many as 25 different work centers. The material handling system is divided into the primary and secondary systems. The primary system moves parts between work centers, whereas the secondary system moves parts within the work centers. The computer control system is emulated by specifying dispatching and part selection rules in the system. **FMS** links the descriptions of the four flexible manufacturing components into a model to gather information about system performance.

CHAPTER IV

MODEL FORMULATION IN **FMS**

Introduction

The purpose of this chapter is to present the method for translating the system description of a flexible manufacturing system into an executable model using the **FMS** language. In addition, this chapter will first provide an overview of the functional relationships of the **FMS** language. It will also include the description of the model editing capabilities of **FMS**. The language overview presented in this chapter will serve as a reference point for Chapter V: Model Execution and System Reporting.

Language Overview

The **FMS** simulation language can be divided into six main modules as shown by the **FMS** Main Menu in Figure 2. This chapter will discuss modules 1, 2, 5, and 6 shown on the main menu. Modules 3 and 4 will be discussed in Chapter V.

Model Control Parameters

Module 1 can be subdivided into three smaller modules as shown by the menu in Figure 3. The module titled Header

```
**************************************************
*                                                *
*      FLEXIBLE MANUFACTURING SIMULATOR          *
*                                                *
*                  Main  Menu                    *
*                                                *
*  Option            Function                    *
*                                                *
*    1.      Enter model control parameters      *
*                                                *
*    2.      Enter model statements              *
*                                                *
*    3.      Execute  model statements           *
*                                                *
*    4.      Review  output reports              *
*                                                *
*    5.      Edit control parameters             *
*                                                *
*    6.      Edit model statements               *
*                                                *
*    7.      Terminate the program               *
*                                                *
**************************************************
Enter the option corresponding to the function you wish to execute
```

Figure 2. **FMS** Main Menu

```
**********************************************************************
*                                                                    *
*          MODEL CONTROL  INFORMATION                                *
*                                                                    *
*    Option                    Parameters                            *
*                                                                    *
*      1.            Header and Project Information                  *
*      2.            Part  Information                               *
*      3.            Statistics Gathering Information                *
*      4.            Return                                          *
*                                                                    *
**********************************************************************
Please enter your option: which option ?
```

Figure 3. Model Control Sub-Modules

and Project Information is where the user can attach labels to a specific model such as the project title and modeler's name. The Header and Project Information module also contains a sub-module entitled Model Execution Parameters. In this module, the user would enter the starting and ending simulation times along with the start and stop times for the TRACE option and the time to clear statistics. Also included in this module are the specifications for the maximum number of parts allowed in the system and the maximum number of material handling carriers available in the model.

The Part Information module is where all of the part parameters such as processing times, dispatching rules, and routings are entered. The third module entitled Statistics Gathering Information is where the user may select one or more of the summary reports to be formatted at the end of the simulation run.

Model Statement Modules

Module 2 in **FMS**, Enter Model Statements, contains the actual system description, whereas module 1 provides inputs to the model. The model statement module can be divided into four sub-modules as shown in Figure 4. The work center descriptions involve specifying the name, type, and capacity of a work center. The shuttle descriptions are used to provide a load and unload mechanism for the work centers. Also, the shuttle descriptions link the part queues to their

respective work centers. The material handling description provides the model with the characteristics for the primary workpiece movement system, whether it is a system of wire guided carts or an overhead conveyor. The breakdown descriptions provide information concerning the frequency, duration, and causes for failures at work centers.

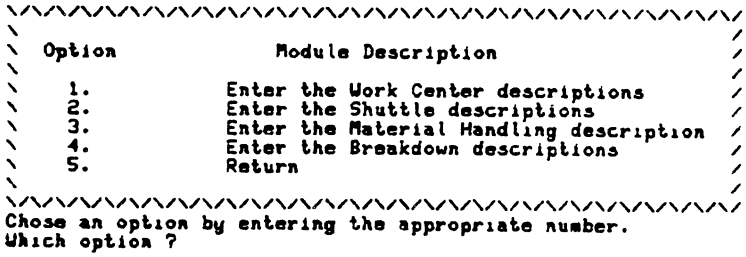


Figure 4. Model Statement Modules

Execution of Simulation Module

The third module of **FMS**, Execute Model Statements, is where the actual simulation is carried out. The model statements and the control parameters are processed through an eight stage error checking routine. If critical errors are detected then no attempt is made to execute the model. In the other case where zero critical errors are detected, then **FMS** builds an internal list of pointers to form the model elements into a network. After the network building phase the filing system, the statistics structures, and the

model are initialized to conduct the simulation. Finally, the simulation is executed after the initialization phase. When errors are encountered, whether minor or critical, **FMS** pages the error messages on the terminal screen and then pauses for the user to read them. Once all error messages have been displayed the model will continue, in the case of minor errors, or will return to the main menu, in the case of critical errors.

Reviewing the Output Reports

In this module the user will view the statistics that were collected from the simulation in a set of formatted screens according to the reports selected in module 1. If the user desires to have one or more reports saved in a file to obtain a hard copy, he or she must simply answer YES to the save question before moving to the next report.

Editing the Control Parameters and Model Statements

Modules 5 and 6 contain routines to make changes in each of the sub-modules contained in the first two main modules of **FMS**. Each of the routines for editing modules is menu driven, thereby relieving the user of memorizing numerous commands. The main menus for the two edit modules, Control Parameters and Model Statements, are presented in Figure 5 and 6.

1 General Project Titles

2 Model Execution Parameters

3 Part Information

4 Exit

Which module would you like to edit?

Figure 5. Edit Control Parameter Menu

1 EDIT WORK CENTERS

2 EDIT SHUTTLES

3 EDIT MATERIAL HANDLING PATHS

4 EXIT

Which module do you wish to edit?

Figure 6. Edit Model Statement Menu

Description of Control Parameter Input

General Project Titles

The input module entitled General Project Titles contains three input parameters. The three parameters are the project title, the modeler's name, and the current date. All three of these items are treated as character strings by **FMS.** The reasoning behind this was to allow the user the freedom to enter any combination of ASCII characters to form a label. As is the case with other languages, titles are restricted to a small set of characters when forming model labels. The length of the labels in **FMS** is limited to strings less than 80 characters.

These three input items are included in the software as a convenience to the user for the purposes of model identification. They are not essential to creating or executing a model with **FMS.** Consequently, no error checking is performed on these parameters and they may be omitted when creating models.

Model Execution Parameters

The set of model execution parameters consists of five items: the starting and ending clock times, the starting and ending times for the TRACE option, the time to clear statistics, the maximum number of parts allowed in the system, and the maximum number of material handling carriers available. The starting and ending clock times, the maximum

number of parts allowed in the system, and the maximum number of carriers are essential to model creation. If any one of the three sets of parameters is omitted, a critical error will result. In the case of a model that does not contain a material handling system, the maximum number of carriers would be zero.

The parameters in this section are all numeric; consequently, when the values are input they must contain only numerals and possibly a decimal point. When the user is asked to input the values, the variables are initially read as character strings and are then converted to the numeric values one character at a time. If an invalid character is encountered, processing on the input is stopped and an error message is generated. The user is then asked to re-enter the value. Cases of invalid characters would be two decimal points, a letter or symbol, or a minus sign. The set of model execution parameters with their constraints is presented in Table VI.

The bounds on the starting and ending times for the TRACE option, and the time for clearing statistics are dependent upon the starting and ending clock times. Consequently, the first item the user is asked to enter in the module is the starting and ending clock times. This allows the validity of the other times to be verified immediately after performing the syntactical check. In the case where the value entered is too large for the system, an

error message is generated and the user is asked to re-enter the value.

TABLE VI

BOUNDS ON MODEL EXECUTION PARAMETERS

Lower Bound	Parameter Name	Upper Bound
0.0	Beginning clock time	Ending clock time
Beginning clock time	Ending clock time	2^{31}
Beginning clock time	Start of trace	End of trace
Start of trace	End of trace	Ending clock time
Beginning clock time	Time to clear statistics	Ending clock time
0.0	Maximum number of parts in the system	2^{15}
0.0	Number of available carriers	2^{15}

Part Descriptions

In this section information concerning the number of different parts and their associated process plans and production schedules is formulated. The modeler is allowed

to enter information concerning from 1 to 20 parts. Associated with each part is a part name, an operation set, and an operation sequence. The part name may be any sequence or combination of letters, numbers, symbols, and blanks less than 31 characters in length. This should allow the user a great deal of flexibility in customizing the part information.

The operation set may contain from 1 to 50 different operations. An operation is defined to be any activity that has a non-negative duration and takes place at a work center. The exception to this definition is the loading of a part into the system, which is performed automatically by **FMS**. As an example, assume there is a system with three work centers excluding the load station. Also, assume that a part, named PART-ONE, needs to be processed at stations one, two, three, and then at station two again. PART-ONE would then have four operations in its operation set, since it visits four work centers with each operation consuming an amount of time greater than zero.

The operation set for a part will contain additional information about each operation. Specifically, the operation set will contain a processing distribution for each operation, a dispatching rule, and a list of work centers where the operation can be performed. The possible distributions for processing times are presented in Table VII. The dispatching rules that are available in **FMS** are presented in Table VIII. These dispatching rules are used

TABLE VII

AVAILABLE STATISTICAL DISTRIBUTIONS

Distribution Name	Required Parameters
Exponential	Mean, standard deviation
Normal	Mean, standard deviation
Lognormal	Mean, standard deviation
Uniform	Upper and lower limits
Triangular	Lower limit, mode, upper limit
Constant	Any constant

TABLE VIII

AVAILABLE DISPATCHING RULES

Rule	Definition
SNIQ	Select the machine with the smallest number of parts in the queue(s).
SMHIT	Select the machine with the largest amount of idle time.
Random	Select a machine at random.
LAPT	Select the machine that has the least amount of processing time in the queue(s).

to select a work center from the list of feasible work centers for an operation. The work center list contains from 1 to 10 work center names where an operation can be performed. The syntax rules for work center names will be discussed in a later section of this chapter. The combination of dispatching rules and feasible work centers will form the basis for part routings. The operation sequence will serve as the third parameter in determining the routing sequence for a part. The operation sequence is a linear list of operation numbers arranged in the order that the operations are to be performed. Consequently, when **FMS** is given a set of ordered operations and work centers where each operation can be performed, a finite set of routes through the system can be enumerated. However, with the use of dispatching rules, the path or route a part follows through the system will be dynamically determined at each step in the operation sequence. **FMS** will find the next operation to be processed and then select the work center where the operation is to be performed using the dispatching rule that was specified by the modeler. **FMS** will then determine how to move the part from the current work center to the next work center via a shuttle or material handling path. If a link to the next work center cannot be found, **FMS** will select a new work center disregarding the first one selected. It would be impossible for **FMS** not to find at least one feasible work center, since this condition is tested for in the error checking modules before execution.

The manner in which **FMS** tests for this error will be explained in Chapter V.

Part Description Input Sequence. The part description input sequence begins by asking the user if he or she would like to see the instruction set. If the user answers "YES" then the instructions are displayed on the user's terminal screen. Once the instructions are printed or if an answer of "NO" was entered, the user will be asked to enter a part name. Next the user will be asked how many operations there are for this particular part. After specifying the number of operations the user will be prompted to enter the processing distribution and the dispatching rule, separating the parameters by a comma. If either parameter is invalid the user will be informed of this and asked to re-enter the invalid parameter(s). Once the distribution has been verified by the system, the user will be asked to enter the appropriate numeric descriptors for the distribution. For example, if the user specified the Normal distribution, then the system would prompt for the mean and standard deviation.

The next step is to ask the user to enter the valid work centers for the operation one at a time. When the user is through entering work center names, he or she must type "STOP" to terminate the input sequence. **FMS** will then prompt the user for information concerning the remaining operations.

The last segment in the part description input sequence is to specify the operation sequence. The user will be

given a short example on what information is needed in this segment. Then the user will be asked to enter the operation numbers in the order that they are to be processed as the part moves through the flexible manufacturing system. Finally, after the operation sequence has been entered for the current part name, the user will be asked to enter a new part name or "QUIT". If a new part name is entered then the cycle will repeat itself; if "QUIT" is entered the user will be asked if there are any changes to be made. The user must respond "YES" or "NO" to this question. A "YES" reply will invoke the Part Editor; otherwise the user will enter the Production Scheduling input sequence.

Production Scheduling Input Description. The production schedule is a list of part names with an associated lot size and input or load frequency. The part name must conform to the syntax rules presented previously and they must also match the part names that exist in the part descriptions. If the user enters an invalid part name, the set of part names currently existing in the part descriptions is displayed on the screen. The lot size must be an integer between 1 and 32,767. If an invalid entry is made the user is asked to re-enter the lot size. If by chance, the user enters a real valued lot size the value will be converted to its integer component and a message will not be generated. The load frequency must be one of the previously mentioned statistical distributions. The

production lots should be entered in the order that they are to be processed.

Production Scheduling Input Sequence. The user is first queried to enter a part name for the first production lot. The validity of the name will be checked and if it is a valid name the user will be prompted to enter the lot size and the input frequency, separating the parameters by a comma. After the validity of both parameters has been checked and found to be legal, the user is prompted for the distribution descriptors. As was previously mentioned, if any parameter is in error, the user will be immediately asked to re-enter the erroneous parameter. After the user has entered the lot size and the distribution name with its descriptors, the input cycle repeats itself by asking for a new part name. When the user wants to terminate the input sequence he or she must type the word "QUIT".

Statistics Gathering Information. In this input segment the user is asked which, if any, of the four types of reports should be formatted at the end of the simulation run. The four types of reports are as follows:

1. Production Summary Report
2. Utilization Summary Report
3. Queue Summary Report
4. Breakdown Summary Report

The production report includes the work in process information. Also, the utilization report actually consists

of three summary reports for work centers, shuttles, and the material handling system.

The advantage of selecting the summary reports in advance is that as soon as the simulation run is complete, the required statistics are stored in a memory resident buffer. Consequently, when the output facility is entered the time to format the reports on the screen is reduced. This time reduction will probably not be noticed during slack usage periods. However, the user will not notice much, if any, delay in formatting the reports during peak computer usage periods.

A second advantage to the statistics collection procedures in **FMS** is that just because a report is not selected in this section does not mean the modeler cannot request it later in the output facility. This option is possible since all relevant statistical information is collected and stored during the model execution phase. However, only the requested statistics will be computed prior to entering the output facility. The implication of this feature is that the modeler does not need to execute this module in order to run a model.

Description of Model Statement Input

Work Center Descriptions

The work center descriptions consist of three descriptors per work center which fully describe it. The three descriptors are the work center name, the capacity of

the work center, and the work center type. The work center name can consist of any combination or sequence of letters, symbols, numbers, or blanks. If a name greater than 30 characters is entered it will be truncated and the user will not be notified. The ability of the modeler to create names for individual work centers provides both flexibility and uniqueness to any model. The flexibility is realized by the modeler not having to remember rigid work center types or naming conventions. The uniqueness of a model is possible since an almost infinite number of models could be created without any work center names being duplicated. An additional advantage to allowing user named work centers is that a model consisting entirely of customized work center names will be easier to communicate to other personnel, both technical and non-technical.

The capacity of a work center is defined as the number of parts that can be processed in parallel by the work center. The capacity of any given work center is limited to 25 parts. This limit does not restrict the language unreasonably for developmental purposes; however, a commercial version of **FMS** would probably need to allow at least 50 parallel processors per work center.

The work center types can be classified into one of three general categories: a load station, an unload station, and a regular work center. In **FMS** the modeler must include one load station and one unload station. If either one or both of these two types are omitted in the work

center descriptions an error will result. The modeler must then modify the current work center descriptions or add a new work center to alleviate the deficiencies. The bulk of the work center descriptions will be regular work centers. The reason for classifying work centers into the three types was to provide a structure for specifying the system input (a load station), the system output (an unload station), and the system work centers (regular work centers).

Work Center Descriptions Input Sequence. The work center descriptions may be entered in one of two modes. In the first mode the modeler may create a new set of work center descriptions by typing the information in via a terminal. In the second mode the modeler may read a specific file where a set of existing descriptions resides. The modeler may choose either mode by replying to the question: "Would you like to use NEW or EXISTING work center descriptions?" The replies are either "NEW" to create new descriptions or "OLD" to use existing descriptions. If the "OLD" option is chosen the **FMS** Filing System is invoked. The **FMS** Filing System will be discussed later in this chapter.

The "NEW" option allows the modeler to create a set of work center descriptions containing from 2 to 25 work centers. However, before the modeler begins creating the descriptions there are two questions requiring replies which the modeler will be asked. First, the modeler will be asked

if he or she would like to see the instruction set. The valid responses are "YES" and "NO". A response of "YES" will cause the instruction set to be printed on the terminal screen. If a response of "NO" was entered, or after the instructions are finished printing, the second question that will be asked is how many work centers there are in the system. The valid responses to this question are integer values between 2 and 25. If an invalid work center value is entered, the user is informed of this and is asked to re-enter the value.

Once the user has provided an appropriate response to the above questions, the actual work center descriptions can be entered. The user is given a prompt asking for the description of each work center by number beginning with work center one. The order of inputs required is the work center name first, the work center capacity second, and the work center type last. The three parameters should be entered on one line, separating each item by a comma. When the user has completed the input sequence, he or she will be asked if the newly created work center descriptions are to be saved in a file. If they are, then the **FMS** Filing System is called and the user must specify a file name where the descriptions are to be stored. In the case where the descriptions are not to be stored, control of the program is transferred back to the main program.

Shuttle Descriptions

The description of a shuttle is comprised of seven descriptors: name, capacity, queueing rule, queue number, work center link, the shuttle type, and the movement distribution. The shuttle name is subject to the same syntax rules as the work center names. The reasoning behind allowing user defined shuttle names was the same as for the work centers. The capacity of the shuttle is defined as the maximum number of parts that can be transported in a single movement. The maximum allowable capacity is 128 parts, which should provide the ability to model a majority of transfer devices. The method for reading numeric input is the same throughout **FMS** in that all numerical values are first read as character strings and then converted to the numbers if possible. If the conversion is not made, then an error message is issued and the user is asked to re-enter the value. Again the reason for doing this is to avoid all types of overflow and conversion errors.

The queueing rule is the mechanism which will specify how arriving entities will be placed in the queue associated with the shuttle. The available queueing rules are presented in Table IX. The queue number is simply the queue where the shuttle being described will receive or deposit parts. The queue number is a numeric value and must have a value greater than zero and less than 101. The work center link is the name of the work center that is serviced by the shuttle. Again the work center link must follow the syntax

rules established earlier. Should a typographical error occur in the work center link, it will go undetected until the modeler tries to execute the model. At this time the modeler will be informed of an extraneous work center link.

TABLE IX

AVAILABLE QUEUEING RULES

Rule	Definition
FIFO	First in => First out
LIFO	Last in => First out
RAND	Stored at random in the queue
SPT	Shortest processing time first

In **FMS** there are two types of shuttles: input and output. Input shuttles are designated by entering the letter "I" and output shuttles are designated by entering the letter "O". The logic associated with specifying a shuttle to be input or output is determined by the type of relationship that is necessary with the work center link. A shuttle name may appear more than once in the list of descriptions; however, this should only be done when a shuttle serves more than one work center. The movement

distribution for a shuttle can be any one of the valid statistical distributions shown in Table VII. To designate a distribution, the user must enter either the first letter of the distribution or the entire name.

Shuttle Descriptions Input Sequence. The initial segment of the shuttle input sequence is identical to the work center input sequence except for the information contained in the instruction set; consequently, it will not be discussed further. The shuttle input sequence is different from the work center input sequence in that it does not ask how many shuttles there are in the system and proceed sequentially. Instead, the input sequence begins by asking the user to input a shuttle name or the key word "QUIT". If the user enters the key word then the shuttle input sequence is terminated. If the user enters a character string other than the word "QUIT", he is prompted to enter the remaining six descriptors on the next line, separating the parameters by commas. The six descriptors are to be entered in the following order:

1. Shuttle capacity
2. Queueing rule
3. Queue number
4. Work center link
5. Shuttle type
6. Movement distribution

Each of the parameters is examined to determine the validity of the entry. If an invalid entry is encountered,

the user is informed of this and asked to re-enter the corresponding parameter. The queue capacity and the parameters for the distribution selected will be asked for after the validity of the queue number and the distribution name have been verified. This cycle of input will continue until the user has entered 100 shuttle descriptions or the word "QUIT" is entered when the prompt for a new shuttle name appears.

The design logic behind not asking the number of shuttles and performing the input in a linear fashion is due to the fact that there may be more logical shuttles in the system than physical shuttles. An example of this situation would be a robot guided on rails loading three parallel work centers. In the physical sense there is only one robot in the system; however, from the logical viewpoint there are three robots in the system. This illustration also brings to light the need to specify the shuttle as either an input or output device for a work center. The advantage of dividing a shuttle into its logical components allows the user to evaluate what portion of the shuttle's different functions comprise its total utilization.

In **FMS**, the technique used by the modeler to divide the shuttles into their logical functions is to enter one description for every task that is to be performed by the shuttle. This design allows the modeler greater freedom and power in assigning queues and movement distributions. As an example, suppose a robot performed two completely different

tasks. If the two tasks had to be described by one movement distribution, some of the modeling detail would be lost. However, in **FMS** the modeler may break down the tasks and assign a unique movement distribution to each movement.

Material Handling Descriptions

The material handling description is divided into 100 zones or paths. A zone is the basic element in the material handling system; it can be described by four characteristics. These are the starting point, the ending point, the zone distance, and the travel rate. The starting and ending points must be valid work center names and must follow the syntax rules presented earlier. The distance of a zone or the path length must be a positive number less than 2,147,483,648. Finally, the travel rate must be one of the aforementioned statistical distributions.

The structure of the material handling descriptions allows the modeler a great deal of flexibility in modeling various material handling systems. The specification of path lengths and travel rates allows the modeler the ability to adjust and experiment with spatial relationships as well as material flow rates. When the ability to vary the number of available carriers is considered, as previously mentioned, one can begin to grasp the amount of flexibility available when modeling the material handling section of a flexible manufacturing system.

Material Handling Input Sequence. The material handling input sequence begins by first checking to see if the work center descriptions have already been entered. If they have not, then the user is asked to enter the work center descriptions before entering the material handling system. The reason for this is to allow the validity of the starting and ending points for each zone to be checked as soon as they are entered. In the case where the work center descriptions have already been entered, the user is next asked if he or she would like to use "OLD" or "NEW" material handling descriptions. After the proper reply the program either passes into the FMS Filing System or the user is asked if he or she would like to see the instruction set. Upon either viewing the instructions or bypassing them, the user enters the main material handling input loop. This loop starts with the first work center and progresses sequentially through the work center descriptions asking how many paths emanate from the work center. If the user enters a positive valued integer, then a second loop is entered. Inside this inner loop the user specifies the ending point, the path length, and the movement distribution for each path emanating from the work center. Upon termination of this inner loop, control of the input sequence is returned to the main loop, whereupon the work centers are incremented and the loop starts over. If there are not any paths emanating from a given work center, the user must simply enter a zero when prompted.

Once the user has completed the material handling input loop, he or she will be asked if the material handling descriptions are to be saved in a file. If they are, the filing system is invoked; otherwise, control of the program is returned to the model statement input menu.

Breakdown Descriptions

The breakdown description for a work center consists of four items: the work center name, the frequency distribution, the duration distribution, and a textual description of the breakdown. The work center name must follow the same rules as previously discussed. The frequency distribution and the duration distribution must be one of the statistical distributions mentioned earlier. The textual description of the breakdown can be any sequence of ASCII characters up to 400 characters or 5 lines.

The design of **FMS** with respect to breakdowns is such that the user may specify one or more breakdowns for a given work center. The frequency distribution describes the time between occurrences of an individual breakdown. The duration distribution describes the amount of time that an individual occurrence will last. In the case where more than one breakdown description is present for a work center, **FMS** will allow only one to occur at any time. As an example, suppose WORK-CENTER-ONE is subject to random tool failures and scheduled maintenance. **FMS** would not allow a tool failure to occur during scheduled maintenance. If a

work center is already in the "DOWN" state, **FMS** will simply disregard the second breakdown event and reschedule it for a new time. Also, the occurrence of a new breakdown is scheduled after the completion of the breakdown duration. The result of this is that the next occurrence for a tool failure will not be scheduled until the end of the repair event.

Breakdown Description Input Sequence. The breakdown input sequence begins by asking the user if he or she would like to see the input instructions. If the user requests them, they are then printed on the terminal screen; otherwise, the user is prompted for the first breakdown description. The information should be entered with the work center name first, the frequency distribution second, and the duration distribution last. Each of the input parameters should be separated by a comma. After entering the first three parameters, **FMS** performs the usual validity check. If any of the parameters are in error, the user will be asked to re-enter the bad parameter.

The numeric parameters that correspond to the two statistical distributions will be asked for after the distribution names have been verified. Then the user will be asked if he or she would like to input a description of the breakdown. When the prompt appears, the user is allowed to enter up to five lines of text describing the breakdown scenario. If the user desires to end the text input, he or she must type the capital letter "X". After the text

description, the user is prompted for the next breakdown; the cycle will repeat itself until the user enters the word "QUIT" when the new breakdown prompt appears.

Description of Support Routines for Model Formation

The **FMS** Filing System

The **FMS** Filing System is divided into two operating modes: reading files and creating files. In the read mode, the filing system is called to read a file that contains some information concerning a specific module of a model. However, before a file can be read it must exist in the directory. If the user specifies a file name that does not exist, **FMS** will detect the undefined file condition upon execution of the open statement. In the undefined file condition, the filing system will inform the user of this, then print all of the valid file names on the terminal screen. If the file does exist, the filing system will read the header record which designates the contents of the file. If the file contents match the type of descriptions requested, the entire file is read. Otherwise, the user is informed of the type mismatch and the valid file names will be displayed on the screen.

In the create mode, the filing system first opens the file with the appropriate record length and block sizes for the descriptions being stored. The next step is to write a

header record containing a label specifying the type of model descriptors that are contained in the file. Finally, the model descriptions are written into the file and the file is closed. The types of files created by the filing system are keyed sequentially. The key that is used is the record number.

Queue Selection Information

In **FMS** there are two items that describe a queue: the queueing discipline and the queue capacity. The queueing disciplines available in **FMS** were presented in Table IX. The queue capacity must be an integer value between 0 and 32,767 inclusive. Since the queue number and the queue discipline are specified in the shuttle input statements, only the queue capacity will be entered from an external procedure. This external procedure is called from the shuttle input procedure and from the shuttle editor. The user is prompted for the queue capacity which is then checked to ensure that the value entered is indeed legitimate.

In the case where the user references a queue more than once during the shuttle input sequence, he or she is informed of this and asked if any of the existing queue descriptors should be changed. If they are to be modified the user is allowed to enter the data; otherwise, the existing queue descriptors are kept.

FMS Editing Capabilities

In this section the editing capabilities of **FMS** will be presented. The editing features are divided into two sections: one section for editing the control parameters and one for editing the model statements. The two sets of editors will be presented in the following sections and will only be discussed in a general sense. The reason behind this is that within each of the two editor types the logic employed is very similar and straightforward; therefore, a great amount of detail is not necessary to understand the logic.

Editing Control Parameters

The user has the ability to edit the general project titles, the model execution parameters, and the part descriptions. If the user elects to edit the project titles, he or she is given a new menu asking which title is to be changed. The user may then select one or more titles to change by simply entering the new text string when prompted.

The editor for the model execution parameters is not really an editor. Instead of editing the control parameters, the user is asked to re-enter the whole set of parameters. On the surface this redundancy appears to be somewhat inefficient. However, the time required to re-enter the set of parameters is less than one minute. This time is quite small when compared to actually invoking a

full-fledged editor, making the changes, and then saving them. The time required to re-enter the parameters is significantly less than if the user were operating in a batch mode from a card deck and had to find a key punch and re-punch one or more cards.

The part editor allows the user to perform three main operations on the part descriptions. First, the user can add a part to the descriptions. Second, the user can delete a part from the description set, and finally, the user may modify an existing piece of information about a part. The part descriptors that may be modified are the part name, the operation sequence, or a specific operation. Also, an operation may be added or deleted from a part description. The procedure for adding a part is identical to the part description input routine and will not be discussed further. The method for deleting a part is to first ask the user which part name is to be deleted. After the user provides a valid part name, the part name and its associated operations and operation sequence are removed from the descriptions. If the user does not provide a valid part name, all of the currently valid part names are printed on the terminal screen.

The routines used to modify a part description are subsets of the routines used in the part input sequence. The user is again prompted for a valid part name. Once the part name has been verified the user is asked which particular descriptor is to be modified, such as the name, a

specific operation, or the processing time for a specific operation. After the user has selected the specific data item to be modified, he or she will be asked to enter the new data item. **FMS** will then perform the appropriate data verification according to the type of data, and inform the user if there is an error and allow re-entry of the data.

Model Statements Editing

The structure of the model statement editor allows the user to edit the work center descriptions, the shuttle descriptions, and the material handling descriptions. The structure of the individual module editors is identical. The user is allowed to either add an element, delete an element, or modify any one of the descriptors associated with a particular element within a module. In the add mode a check is performed to ensure that there is sufficient space in the particular structure for the new element. If there is not enough space, the user is informed of the condition and told to either delete an element first or simply return to the main menu. In the case where the space is available, the routines used for adding an element are the same as the corresponding module input routines.

The second function in the model statement editors, deleting an element, is very straightforward. The user is asked for a valid element name in the case of the shuttles and work centers, or for the start and end points for a material handling path. Then the existence of the element

is verified. As always, if the user enters an invalid element, he or she is informed of this and is provided with the list of valid elements. Upon entry of a valid element, the deletion from the appropriate descriptions is made.

In the third edit function, modify an element, the user is first asked to provide the shuttle name, the work center name, or the start and end points for a material handling path, depending upon which module is being edited. After the element has been verified as existing, the user is provided with a choice of the possible descriptors of that element which can be modified. The user may then select one or more of the descriptors to modify. When the user has completed the necessary modifications the program will return to the primary edit menu.

Summary

In this chapter the methods for creating a model of a flexible manufacturing system using **FMS** was presented. The techniques used by **FMS** to interpret both the physical and logical relationships were presented along with numeric bounds on the size of the system that can be modeled using **FMS.** To create a model using **FMS** the modeler must analyze the interaction of the shuttles, work centers, and material handling system. **FMS** facilitates this analysis by providing the modeler the ability to construct the functional rules for the model from each element's basic tasks. An example of this collection of basic tasks into a functional rule

would be a shuttle providing output for work center A and input for work center B. Also, the description of material handling paths between points A and B would serve as a collection of tasks defining a functional rule.

This chapter also presented the editing capabilities of **FMS.** Within **FMS** the modeler is allowed to edit the model statements and the execution parameters. In addition, the modeler may edit the part descriptions as necessary. This is an important feature since the part descriptions are a major variable in determining system performance. The information presented in this chapter demonstrates that to create a model in **FMS,** the modeler must have adequate knowledge of the system. However, the level of interface between the modeler and the computer amounts to simply answering questions about the system. The amount of simulation knowledge required is minimal. The only segment where an exposure to simulation would be helpful would be in the execution parameter input module.

CHAPTER V

MODEL EXECUTION AND SYSTEM REPORTING

Introduction

This chapter is divided into four major sections: **FMS** Error Checking and Execution Sequence, **FMS** Output Facility, Language Verification, and Test Models. The error checking and execution section will present the logic of the error routines, along with possible causes and corrections for the associated errors. Also presented in this section is a detailed discussion for the internal workings of the **FMS** control of event routines. The second section in the chapter, **FMS** Output Facility, will discuss how the reports are generated and the options that are available in the output subsystem. The third section, Language Verification, will compare the results obtained from three queueing models to three models developed in **FMS.** Finally, the last section will present four basic models of common flexible manufacturing configurations in order to demonstrate the features of **FMS.**

FMS Error Checking and Execution Sequence

Error Checking

The **FMS** error checking procedure consists of eight stages or steps. Each of the first six steps performs checks on the validity and logic of each input module. The last two steps in the process build the actual network and then test the logic of the entire network. At the end of each step any error messages generated in that step are printed on the user's terminal to be interpreted. Once a full screen of messages has been displayed the user must hit a carriage return to continue. This allows the user to view the error messages leisurely without the program controlling the user. The function of each step will be presented individually in the order that they are performed by **FMS.**

Error Stage I. The first stage in the error checking sequence involves verifying that the work center descriptions contain one load station, one unload station, and at least one regular work center. If the minimum conditions are not satisfied, an error message is issued and the fatal error condition is raised. However, even though the error condition is raised, the error sequence will complete its cycle. If the work center descriptions contain either an extra load or an extra unload station or both, they will be disregarded by **FMS.** In this case the fatal error condition will not be raised.

Error Stage II. The second stage uses the list of work centers from stage one to verify that every work center has an input and an output shuttle. If the user has specified more than one input shuttle or one output shuttle per work center, an error message is generated and the fatal error condition is signaled. Likewise, if a work center is missing either an input or output shuttle or both, a message is issued and the error condition taken.

Error Stage III. The third stage of the error checking verifies that the material handling paths are acceptable. The material handling paths will not pass the error check if one or more paths have start and end points that are equal. If this situation arises, the standard action of issuing a message and signaling the fatal error condition is taken.

Error Stage IV. In the fourth stage of the error checking routine, the elements in the breakdown descriptions are cross referenced with the work center descriptions. In the breakdown descriptions, if the user enters a work center name that is not in the list of work centers developed in stage one, an error message is generated. **FMS** will then ignore the invalid references to this breakdown and exclude it further from the model. The fatal error condition will not be signaled, thereby allowing processing to continue excluding previous errors.

Error Stage V. The fifth stage performs two related checks. First, the model is examined to make sure that the

part descriptions and the production schedule were entered. If they were omitted the user is informed of this condition and the fatal error condition is raised. In the second segment of this stage, each work center list within each operation within the part descriptions is examined to verify that there is at least one valid work center in each list. As usual, if there is not one valid work center, the error condition is raised and an error message generated.

Error Stages VII and VIII. In the final two error stages the actual network used in executing the model is constructed and tested for logic errors. However, if the fatal error condition is set to true in any one of the previous stages, then this step will not be attempted. Instead, **FMS** will bypass this step and proceed directly to the model echo routines and then return to the main program.

The network construction is performed by first determining the size of the network. The network size is the sum of the number of material handling paths, the number of work centers, and the number of shuttles. All other items such as queues, breakdowns, and distribution parameters are maintained outside the network since they provide only control of input to the network. The next step after determining the size of the network is to locate the start and end points or the load and unload stations.

The actual network construction begins by starting at the unload station and performing a backwards pass, similar

to CPM methodology, until the load station is reached. The first step in the backwards pass is to make the unload station the "current" station and to add it to the stack of network elements. Next, all material handling paths that have the "current" station as an end point are traversed to find their start point. As each start point is located, it is added to the "to-be-processed" element stack. After the material handling paths have been evaluated, a search is made to find all work centers linked to the "current" station by a shuttle. All of these stations are also added to the "to-be-processed" stack. As each of the material handling paths and each of the shuttles are processed to find their originators, they are added to the stack of resolved elements. The current step of the backwards pass ends when all incoming material handling paths and shuttle links have been evaluated. The "to-be-processed" stack is queried for the next element to initiate another step in the cycle. If the stack is empty, the loop is terminated and the cross check of the network elements is initiated.

The element cross check consists of comparing all of the elements in the work center structure, in the shuttle structure, and the material handling structure to see if they exist in the network stack. If there are any discrepancies, such as work centers that were not located in the backwards pass, then the error condition is raised and the extraneous elements are listed with possible causes for the problem. The error checking sequence will then either

pass control of the program to the execution sequence, or return to the main procedure to allow the user to correct any model deficiencies.

Model Execution Sequence

The model execution sequence consists of simulating the model description that was received from the error checking stages. In order to perform the simulation, **FMS** must create and process entities with certain attributes. The control program uses ten discrete events to create and process these entities according to the structural and logical relationships defined in the model description. The discrete events have associated with them one or more procedures for queue manipulations, statistics collection, and event calendar manipulations. The **FMS** logic for executing a model will be presented by first defining the entity attributes and how they are interpreted, and secondly discussing each discrete event and its associated logic.

The procedures used for queue manipulations, statistics collection, and event calendar manipulations will be presented. Finally, the logic for executing a model will be traced to summarize the logic flow for executing a model.

Attributes Definition. Each entity or part created during the model execution phase will carry with it seven attributes. These are presented with their definitions in Table X. The mark time, attribute one, is used to calculate the time spent in the system when the part reaches the

unload station. The remaining attributes are used to control the flow and routing of a part through the system.

TABLE X

ATTRIBUTE DEFINITIONS

Attribute Number	Definition and Purpose
1	Mark time or time of creation. Used to calculate time in the system.
2	Work center number for next operation.
3	Value for specifying position in the queue.
4	Work center number of where current operation is to be performed.
5	Current operation number.
6	Duration of operation for the part at the current work center.
7	Number of remaining operations for the part. Provides method for queue ranking on the least number of remaining operations.

Event Definitions. In the **FMS** control routine there are ten discrete events used to evaluate and control the logic of the simulation model. Each of the ten events with their definitions is presented in Table XI. In the control routine, events 1 through 6 and event 9 have external

procedures to evaluate their logic. Events 7 and 8 cause a logical variable to be set to "true" and "false" respectively. The logic of event trace will be presented in a later section. The logic of events 1 through 6 and event 9 will be discussed in detail in the subsequent sections.

TABLE XI

DEFINITIONS OF FMS DISCRETE EVENTS

Event Number	Definition
1	Load a part into the system.
2	End of movement by a shuttle, or an arrival to a work center.
3	End of movement by a material handling carrier, or an arrival to a work carrier.
4	End of processing for a part at a work center.
5	Occurrence of a breakdown.
6	End of breakdown duration or repair time.
7	Set trace flag to true to initiate event trace.
8	Set trace flag to false to end event trace.
9	Clear the statistical arrays.
10	Terminate the simulation routine and return control of the program to the main procedure.

Loading a Part: Entity Creation. The part loading procedures contain six main steps. Associated with each step may be one or more intermediate steps. The first step is to check and verify that there is room in the system for the part. If there is not room, the procedure jumps to step six and schedules the time to load another part. If there is room in the system, step two is executed to see if a part changeover is needed. The part changeover consists of checking for the next production lot and resetting the parts counter to zero. If the next production lot does not exist, then the schedule is repeated starting with lot one. At the completion of the schedule changeover, control is returned to step three.

The third step consists of using the dispatching rule for the first operation to select the first work center where the part will be processed. The fourth step is to determine which method of movement will be used to transport the part from the load station to the first work center - either a shuttle or a material handling carrier. The fifth step will actually determine the disposition of the part by checking the availability of the movement device.

In the case where the part is to be transported by a shuttle, the status of the shuttle is checked to see if it is busy or idle. If the shuttle is idle, the part is loaded onto the shuttle and the arrival at the next work center or the end of the shuttle movement is scheduled. In the case where the part is to be moved by a material handling

carrier, the availability of a carrier is checked. If a carrier is available the part is loaded onto the carrier and the arrival at the work center is scheduled. In case the shuttle is busy or material handling carrier is unavailable, then the part is placed in an output queue and a request is logged into the logic controller for the part to be removed as soon as possible. The **FMS** logic for responding to request for movement devices places a higher priority on removing parts from output queues, then loading parts. The reasoning behind this was that traffic blockage would be minimized by allowing parts to be unloaded to progress through the system.

Event Calendar Manipulations. The event calendar is maintained by two separate routines called NEW-EVENT and UPDATE. The routine NEW-EVENT is called to place an event on the calendar. Conversely, UPDATE is called to advance the simulation to the next event. The value of the simulated time is stored in the variable TIME. The list of scheduled events or event calendar is maintained in a data structure called CALENDAR. Each record in CALENDAR contains four data items for each event: the event time, the event code, the element pointer, and the part attributes. The records within the calendar are arranged in an ordered list such that the first record is always the next event.

The algorithm used for manipulating the calendar is a modified HEAPSORT. The conventional HEAPSORT as presented by Baase (1) arranges records in nonincreasing order. The

algorithm employed within **FMS** will arrange records in nondescending order. The algorithm employed by **FMS** is an in place sort which precludes the use of external storage. Also, the algorithm employed by **FMS** outperforms doubly linked lists which are used by several other languages. The efficiency of the HEAPSORT is on the order of $\log_2 N$ where N is the number of items in the list. The efficiency of the double linked list is on the order of N where N is again the number of elements in the list. This large discrepancy becomes very important when evaluating a complex system.

End of Shuttle Movement. The movement of a shuttle can be classified into two types of movement: input to a work center and output from a work center. When an input shuttle completes movement of a part, the part will either be scheduled for an end of service on the appropriate work center or it will be filed in the input queue if the work center does not have any available capacity. After the part has either been filed or scheduled on a machine, the input shuttle will perform three checks to see if another part transfer is necessary. First, if the current shuttle has any output responsibilities for a work center then the highest priority request will be serviced. This case might arise in the situation where a robot serves as input and output for multiple machines. Requests for a shuttle are assigned priorities with output taking precedence over input and FIFO being the rule within a specific mode of operation

(input or output).

In the second check performed by the shuttle, any incoming material handling paths will be polled for requests to unload a part from a waiting carrier. The part will then be removed from the carrier and scheduled for an end of movement on the current shuttle. The carrier will immediately be freed to respond to outstanding requests. Once the second check has been completed and been bypassed the shuttle will check to see if there are any parts that can be loaded into work centers with available capacity. If all three checks by the shuttle prove negative then the shuttle is set to idle. As the checks are performed corresponding statistics are collected. As an example, if the shuttle removes a part from a carrier then an end of movement is scheduled for the shuttle. The number of available carriers would be incremented causing statistics to be collected on carrier and path utilization. Also, statistics would be collected on the state of the shuttle. A detailed discussion of the **FMS** statistics collection procedures will be presented in a later section.

The end of movement for an output shuttle will cause a very similar chain of events as the end of movement for an input shuttle. First, the disposition of the part must be determined. The current part will either be placed on a material handling carrier, filed in the output queue, or placed in service at the next work center in the operation sequence. The part will only be placed in the output queue

if a carrier is required and one is not available. If the output queue is full then the shuttle will become blocked until a part is removed from the output queue. If the part can travel to the next work center via the output shuttle but the work center is full then the part will be filed in the input queue for the work center.

Once the disposition of the part has been determined then the shuttle will check for any unload requests. Second, it will check for any carriers to be unloaded. Finally, it will check for any load requests. Obviously, the shuttle will only proceed to these checks if it has disposed of the current part.

End of Movement by the Material Handling System. The end of movement by the material handling system is the event of a material handling device, possibly an AGV cart, delivering a part to a work center. Although the part is being delivered to a particular work center the services of the input shuttle for the work center are required. Consequently, the first check is to verify the status of the specific shuttle that is required. If the shuttle is unavailable then the material handling path becomes blocked and a request is logged for the shuttle.

If the shuttle is free then the part is scheduled onto the shuttle and the carrier is freed to reply to any outstanding requests. Finally, relevant statistics for this event are collected before returning to the executive routine.

End of Processing at a Work Center. The end of processing at a work center requires that the part causing the event be transferred to the next work center in the operation sequence or that it exit the system. To determine the disposition of the part the number of completed operations is incremented [Attribute (5)]. The next work center is located by calling subroutine INCOPS. Subroutine INCOPS uses the dispatching rule and the list of feasible work centers for the operation number whose value is stored in Attribute (5) to determine where the part is to be routed. The location of the next work center is stored in Attribute (2). The processing time for the next operation is updated and stored in Attribute (6). The next task is to check the output shuttle. If the shuttle is busy then the part is put in a request file and the work center becomes blocked. If the shuttle is free then the part is scheduled onto the output shuttle.

The final step in the end of processing routine is to check for the existence of more parts awaiting processing at the current work center. If there are parts available and the shuttle is free then the first part is scheduled onto the input shuttle. This final step is only performed if the work center had not previously become blocked.

Breakdown Occurrence. The occurrence of a breakdown at a work center will cause all parts currently being processed to be stopped until the breakdown is over. The method used for altering the times for these parts is as follows.

First, a duplicate structure of the event calendar is created. The second step is to go through a loop once for every event currently on the calendar. As this loop is processed the first event on the true calendar is removed. At this time a check is made to see if this event needs to be delayed until the end of the breakdown. The event is then filed onto the duplicate calendar. The result of this step is that at the end of the loop the duplicate calendar will contain all events with the effects of the breakdown. The third step is to unstack or remove the events from the duplicate calendar and file them back into the true calendar. Finally, the last step is to set the work center to "down" and collect relevant statistics.

End_of_Breakdown. The end of breakdown routine performs two tasks. First, the next occurrence of that type breakdown is scheduled. The second task is to set the machine to "up" and collect relevant statistics.

Trace_Events_and_Clear_Statistics. The occurrence of event type seven will cause the trace facility to be enabled. Conversely, event type eight causes the trace facility to be disabled. Internally **FMS** handles this by setting a logical variable to true for event seven and false for event eight. The actual trace routine is called immediately after updating the simulation clock if the trace facility is enabled.

The trace facility prints the event code, the element

pointer, all part attributes, and the current value of TIME. The output is double spaced and placed 25 lines per page.

An event type nine resets the statistical collection arrays to zero. The routine that is used to do this is the same routine that is called by INITIALIZE at the beginning of the simulation. The time of statistics clearing is reset to the current value of TIME.

FMS Queue Manipulations

In **FMS** there are three structures used to maintain entities in queues. The first structure contains the ranking criteria, the maximum size, and a pointer to the head of each queue. The other two structures are closely related since they are where the queued parts reside. The second structure is the FILES structure and the third is the SPACE structure. Both the FILES and SPACE structures are dimensioned to "partmax." Each node within the structures is made up of three data items. The three data items are the "key" value, the attributes list, and the "link." The link points to the successor node in the queue. The key value is how the entities are inserted in a specific queue.

Internally **FMS** ranks each queue or file based on the lowest value of key first. Consequently, before an entity can be filed a key value must be obtained. A queue that is ranked on FIFO uses the time of arrival to the queue as the key. The LIFO rule is the negative of FIFO. To accomodate the SPT rule Attribute (6) is used as the key. Finally, the

RAND rule is handled by generating a random number on the interval U[0,1]. The advantage of manipulating the queues in this fashion is that all queues are physically treated as stacks. Also, the space structure is physically treated as a stack. Algorithms for manipulating stacks are readily available. Although the efficiency of manipulating a stack is not that of a binary tree or a hash table, this application does not warrant a more complex data structure. The reason being that in an **FMS** there will not be a large number of parts in the system due to the desire to minimize work in process.

FMS Statistics Collection

All statistics were collected in one routine and maintained in a structure for each statistics classification. A statistical classification might be work center statistics or queue statistics. In total there are 12 statistics classes. When the collection routine is called it will be passed a code corresponding to the class of the statistic to be updated. Also, the routine will be passed the element pointer and the new value. The element link is used to identify the correct work center or shuttle. A call is placed to the collection routine every time an item such as utilization or down time is encountered in an assignment statement.

The statistics collected within **FMS** are all time average statistics except in a couple of instances. The

exceptions are statistics based upon observation and are average breakdown duration, the time a part spends in the system, and the average production rate.

FMS Output Facility

The **FMS** Output Facility consists of six separate reporting modules. The primary menu for selecting the individual report modules is presented in Figure 7. The user may select any one of the six reports to review. However, as previously mentioned, if the statistics had not been requested previously in the Header/Project Information module then the formatting of the report will be slowed. Each report consists of one or more pages depending upon the amount of information present. The user is presented with the report one page at a time. After the user has finished reviewing a particular page then he or she must hit a carriage return to continue. When the entire report has been displayed the user is asked if the report should be saved in a file. The user may obtain a hard copy of all of the reports that were saved by typing in the command REPORTS after exiting the **FMS** program. Each of the six types of reports available in **FMS** will be presented in the following sections.

Work Center Utilization Summary Report

The work center utilization report provides information concerning the percent utilization, percent blockage, and

percent downtime for each work center. A work center will experience blockage when it is prevented from unloading a part due to either a full queue or the output shuttle being unavailable. The downtime statistics will reflect the occurrence of machine failures in the model. The percent utilization statistics will be a multiple of the work center capacity. To obtain the actual percent of time the work center was busy the average utilization should be divided by the capacity. Each of the three statistics will provide the mean utilization and the variance. An example of the work center utilization summary is presented in Figure 8.

SYSTEM REPORTING

Option	Report
1	Work Center
2	Shuttle Utilization Report
3	Material Handling Utilization Report
4	Work in Process and Production Report
5	Queue Summary Report
6	Breakdown Summary Report
7	Exit

Which report would you like to see?

Figure 7. **FMS** Output Menu

Work Center Utilization Report

Work Center : LOAD

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : DRILL1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.38	0.237	0.00	0.000	0.00	0.000

Work Center : MILL1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.61	0.237	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Figure 8. Work Center Utilization Report

Work in Process and Production Summary Report

The work in process (WIP) and production report is divided into three sections. The first two sections present information on the overall system work in process and system production. The third section provides production information by individual part types. Sections one and two are presented on the first page of the report. The individual part production information is presented on page two.

The work in process information consists of the average number of parts in the system and the associated variance. The production information for the system consists of the average time a part spends in the system and the variance. Also, the mean and variance of the part production per unit

of time is presented in the second section. The individual part production information consists of the average time in the system, the variance, and the number of parts produced. A sample WIP and production report is shown in Figure 9.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 3.51
Variance = 0.33

System Production

Average Time in the System = 15.904
Variance = 0.158
Part Production per unit time = 0.062
Variance = 0.000007

Production by Part Type

Part name	Avg Tis	Var	Number
PART1	15.904	6.305	175.000

Figure 9. Production and Work in Process Report

Shuttle Utilization Summary Report

The shuttle report provides information on the mean and variance of the percent of time the shuttle was busy and the percent of time it was blocked. A shuttle will experience blocking when the queue is full. In addition to the statistics the report designates which queue a shuttle is

linked to and its type. The shuttle statistics are presented with 20 shuttles per page. An example of the Shuttle Utilization Report is presented in Figure 10.

Shuttle Utilization Report

Shuttle Name	%Utilization Mean	%Utilization Variance	%Blocked Mean	%Blocked Variance	Que Num	Type
EX1	0.00	0.000	0.00	0.000	1	O
EX2	0.11	0.099	0.00	0.000	2	I
EX3	0.11	0.098	0.00	0.000	3	O
EX4	0.11	0.098	0.00	0.000	4	I
EX5	0.11	0.098	0.00	0.000	5	O
EX6	0.11	0.098	0.00	0.000	6	I
EX7	0.11	0.097	0.00	0.000	7	O
EX8	0.11	0.097	0.00	0.000	8	I
EX9	0.11	0.097	0.00	0.000	9	O
EX10	0.11	0.097	0.00	0.000	10	I

Figure 10. Shuttle Utilization Report

Material Handling Utilization Report

The material handling report is divided into two sections. The first section provides the mean and variance for the utilization of each path in the material handling system. This group of statistics is useful when evaluating the load on the material handling system between two work centers. The second section of the report provides statistics on the utilization of the carriers in the material handling system. The carrier utilization statistics that are provided are the mean number of carries utilized and the variance. The material handling report is provided in Figure 11.

Material Handling Utilization Report

PATH NUM	MEAN %UTIL	VARIANCE	PATH NUM	MEAN %UTIL	VARIANCE
1	0.11	0.098	2	0.04	0.042
3	0.04	0.042	4	0.04	0.042
5	0.10	0.097	6	0.00	0.000

CARRIER UTILIZATION

Number of Carriers	Average %Utilization	Variance
2	0.35	0.309

Figure 11. Material Handling Utilization Report

Queue Summary Report

The queue summary report provides four statistics for each queue in the model: the mean and variance on the queue size, the minimum, and maximum queue size. The queue statistics are presented with 20 sets of statistics per page. An example of the queue report is provided in Figure 12.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	0.00	0.000	0	0
2	0.00	0.000	0	0
3	0.00	0.000	0	0
4	0.00	0.000	0	0
5	0.00	0.000	0	0
6	0.00	0.000	0	0
7	0.00	0.000	0	0
8	0.00	0.000	0	0
9	0.00	0.000	0	0
10	0.00	0.000	0	0

Figure 12. Queue Summary Report

Breakdown Summary Report

The breakdown summary report provides two statistics for each breakdown. The first statistic that is provided is the average duration for each breakdown. The second statistic that is provided is the frequency of occurrence of that breakdown with respect to time. The breakdown report is shown in Figure 13.

Breakdown Summary Report

Breakdown Number	Element	Average Downtime	Frequency per unit time
1	MACHINE1	9.027	40.000
2	MACHINE2	14.072	133.333
3	MACHINE3	16.015	30.000
4	MACHINE4	13.795	133.333

Figure 13. Breakdown Summary Report

Language Verification

In this section three simple queueing models developed in **FMS** will be compared to the same queueing models developed in SLAM (Simulation Language for Alternative Modeling). The purpose of this comparison is to show that models developed in **FMS** will produce reasonable results when compared to the results from models in another language.

The comparison is not intended to be a measure of superiority of one language over another since the modeling purposes of SLAM and **FMS** are different.

The three queueing systems that are to be used as test models are a single server with an infinite queue, a set of three parallel servers with a single infinite queue, and a set of tandem or serial servers each with an infinite queue. The interarrival and service distributions will all follow a Markovian process. Each of the three test models were solved analytically to provide a baseline for comparing the results from the **FMS** and SLAM simulation models.

Test Model I

The first queueing model consisted of a single server with an infinite queue size as previously mentioned. Also, the system was assumed to have an infinite calling source. The mean interarrival time was exponentially distributed with a mean of 1.20 minutes. The mean service time was set at 0.75 minutes and was also exponentially distributed. The measures of performance used for comparing the analytical model and the two simulation models were as follows: average server utilization, expected queue length, expected number of entities in the system, and expected time in the system. The values for the measures of performance from the analytical model are presented in Table XII.

TABLE XII

TEST MODEL I ANALYTICAL RESULTS

Server Utilization	Number in System	Expected Queue Size	Expected Time in the System
.625	1.67	1.04	2.00

The two simulation models were each run three times for 400 minutes per run. Also, different random number streams were used for each run. The results from the **FMS** model are presented in Table XIII and the SLAM results are presented in Table XIV. Also, the summary reports from SLAM and **FMS** are presented in Appendix A.

TABLE XIII

TEST MODEL I **FMS** RESULTS

Run Number	Server Utilization	Number in System	Avg. Queue Size	Avg. Time in System
1	0.68	1.84	1.16	2.153
2	0.61	1.47	0.86	1.919
3	0.68	1.73	1.05	2.016
Average	0.66	1.68	1.02	2.029
% Difference	4.96	0.77	1.73	1.45

TABLE XIV

TEST MODEL I SLAM RESULTS

Run Number	Server Utilization	Number in System	Avg. Queue Size	Avg. Time in System
1	0.643	1.478	0.835	1.676
2	0.564	1.23	0.666	1.555
3	0.667	2.093	1.426	2.513
Average	0.648	1.600	0.976	1.915
% Difference	3.68	4.01	15.86	4.25

The results indicate that both of the methods performed reasonably well in approximating the analytical model. The results would probably improve with longer run times and a better selection of random number streams. However, the results show that **FMS** performed very well in comparison to the analytical model. The **FMS** results indicate that the language is capable of producing valid estimates of Test Model I.

Test Model II

The second test model involved expanding the service channels to allow three parallel servers. The queue size was allowed to be limitless as was the calling source. The mean interarrival time was changed to 0.444 minutes and the mean service time was changed to 1.0 minutes per entity. The arrival and service processes were still assumed to be

Markovian. The results of the analytical case are presented in Table XV. The two simulation models were both run in groups of three with different random number streams selected for each run. The results of the **FMS** model and the SLAM model are presented in Tables XVI and XVII.

TABLE XV

TEST MODEL II ANALYTICAL RESULTS

Server Utilization	Number in System	Expected Queue Size	Expected Time in the System
2.25	6.0	2.25	8.0

TABLE XVI

TEST MODEL II **FMS** RESULTS

Run Number	Server Utilization	Number in System	Expected Queue Size	Expected Time in System
1	2.42	4.65	2.24	2.016
2	2.24	3.29	1.04	1.435
3	2.29	3.63	1.34	1.542
Average	2.32	3.86	1.54	1.664
% Difference	3.1	14.15	4.55	7.4

TABLE XVII

TEST MODEL II SLAM RESULTS

Run Number	Server Utilization	Number in System	Expected Queue Size	Expected Time in System
1	2.209	3.370	1.609	1.575
2	2.250	4.630	2.377	2.108
3	2.312	3.827	1.514	1.572
Average	2.256	3.942	1.833	1.752
% Difference	0.26	2.17	2.50	2.52

In the second test case **FMS** did not perform as well as in the first case. However, the results are still within acceptable differences. It appears that SLAM and **FMS** traded approximation capabilities in simulating the second test model. The difference in the outputs of both SLAM and **FMS** on the first and second test models could be attributed to random number streams. The reports from the second test model are in Appendix B.

Test Model III

The third test model involved extrapolating the first model to include a second queue and server in series with the first queue and server. The measures of performance for this test will be somewhat different. The comparisons will be based on the utilization of both servers, the size of both queues, and the time spent in the system. The

analytical solution to the tandem queueing system is presented in Table XVIII. The mean interarrival rate was set at 0.667 minutes. The mean service rate for both servers was 0.5 minutes and was exponentially distributed.

TABLE XVIII

TEST MODEL III ANALYTICAL RESULTS

Server Utilization 1	2	Average Queue Size 1	2	Time Spent in System
0.75	0.75	2.25	2.25	3.37

TABLE XIX

TEST MODEL III **FMS** RESULTS

Run Number	Server Utilization 1	2	Average Queue Size 1	2	Time in System
1	0.76	0.74	1.87	1.44	3.04
2	0.76	0.82	2.71	3.88	8.11
3	0.75	0.78	3.20	2.30	4.79
Average	0.757	0.78	2.59	2.51	5.32
% Difference	0.89	4.9	15.25	11.7	33.6

TABLE XX

TEST MODEL III SLAM RESULTS

Run Number	Server Utilization 1	2	Average Queue Size 1	2	Time in System
1	0.777	0.811	2.611	3.025	4.465
2	0.784	0.808	1.917	2.879	4.036
3	0.675	0.627	1.054	0.947	2.342
Average	0.745	0.749	1.86	2.28	3.614
% Difference	0.64	0.13	17.33	1.33	7.24

The **FMS** and SLAM simulation models were run three times a piece similar to the first two test models. The output of the simulation models was very similar. Both **FMS** and SLAM estimated the server utilizations well, with the SLAM estimates somewhat better than **FMS.** Conversely, neither the **FMS** model nor the SLAM model estimated the average queue sizes or the time spent in the system very well; except in the case of the size of queue two, SLAM estimated this measure reasonably well. The large differences between the simulation models and the analytical model can be attributed to two factors. First, the length of the simulation runs was probably not long enough to overcome the initial bias caused by starting the run. This problem could be alleviated by either clearing statistics or making longer runs. However, the purpose of the comparisons was to show that **FMS** will yield approximations similar to another

language. The second factor that could be causing the large differences is the random number streams. The third test model results are in Appendix C.

Conclusions from Test Models

The three test models show that **FMS** will perform similar to another more established language. Also, the tests showed that as the system complexity increases the ability of the simulation model to approximate it will be hindered. The hindrance can be alleviated by clearing transient statistics and making longer runs.

FMS Example Models

In this section four examples of models developed using **FMS** will be presented. The four models will be divided into two pairs with each pair describing two different systems. Each model was formulated to illustrate specific features of **FMS**. The models do not represent actual flexible manufacturing systems; however, they do represent reasonable substitutes. Also, the presentation of the examples is not oriented towards system analysis, consequently each model will be run once for the equivalent of one day of operation. The parameters used for processing and movement times are fictitional.

System I

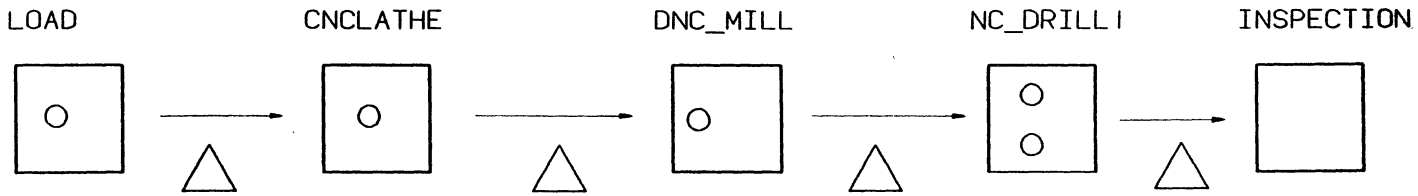
System Description. The first system consists of four

work centers linked by four transfer devices. Also, there are four queues between the work centers. A schematic of the system is presented in Figure 14. The load station is a dummy station and does not generate statistics. The system will process one part type for the first model and will be expanded to two parts for the second model.

Model I. The first model is focused at emphasizing two language features: the flexibility of naming conventions and the modeling of shuttle linkages. The flexibility of naming conventions can be viewed by looking at the input screens which are presented in Section I of Appendix E. The capability of modeling shuttle links is illustrated by having a shuttle provide output functions for one work center and input functions for another. In a more complex system any given shuttle could have more than two tasks associated with two or more work centers. This is an example of how a shuttle is modeled according to its logical tasks.

The output from the first model is provided in Section II of Appendix E.

Model II. The second model of System I was changed to illustrate how more than one type of part can be described to the model. Also, the manipulation of the production schedule to process the parts is a feature of **FMS** that is demonstrated. The production schedule and new part descriptions are contained in Section I of Appendix F. The



KEY

○ = MACHINE SPINDLES

△ = IN PROCESS QUEUES

⟶ = SHUTTLES

Figure 14. Example System I

production schedule contains a part named CHANGE_OVER with a lot size of one. This is not really a part but, rather an entity that will pass through the system to emulate a changeover time for tooling and software. The purpose of this utility is to allow a lag in the system to represent preparation for the new part batch. The output reports for Model II are presented in Section II of Appendix F.

System II

System Description. The second system to be modeled will consist of six work centers. All six work centers will be connected via an oval AGV system. There will be three types of parts that are processed by the system. Two of the parts will require processing at all of the work centers and the third part will require processing at only three of the work centers. Also, a tool change entity will be required for every 50 parts at the NC_DRILL station. A schematic diagram of System II is presented in Figure 15.

Model III. The third example will be a model of System II. The primary feature of **FMS** presented in this model is the emulation of an AGVS within a flexible manufacturing system. Intrinsic to this feature is the capability to specify the number of carts and to vary their travel rate along individual paths. The input screens are presented in Section I of Appendix G and the output from the model is in Section II of Appendix G.

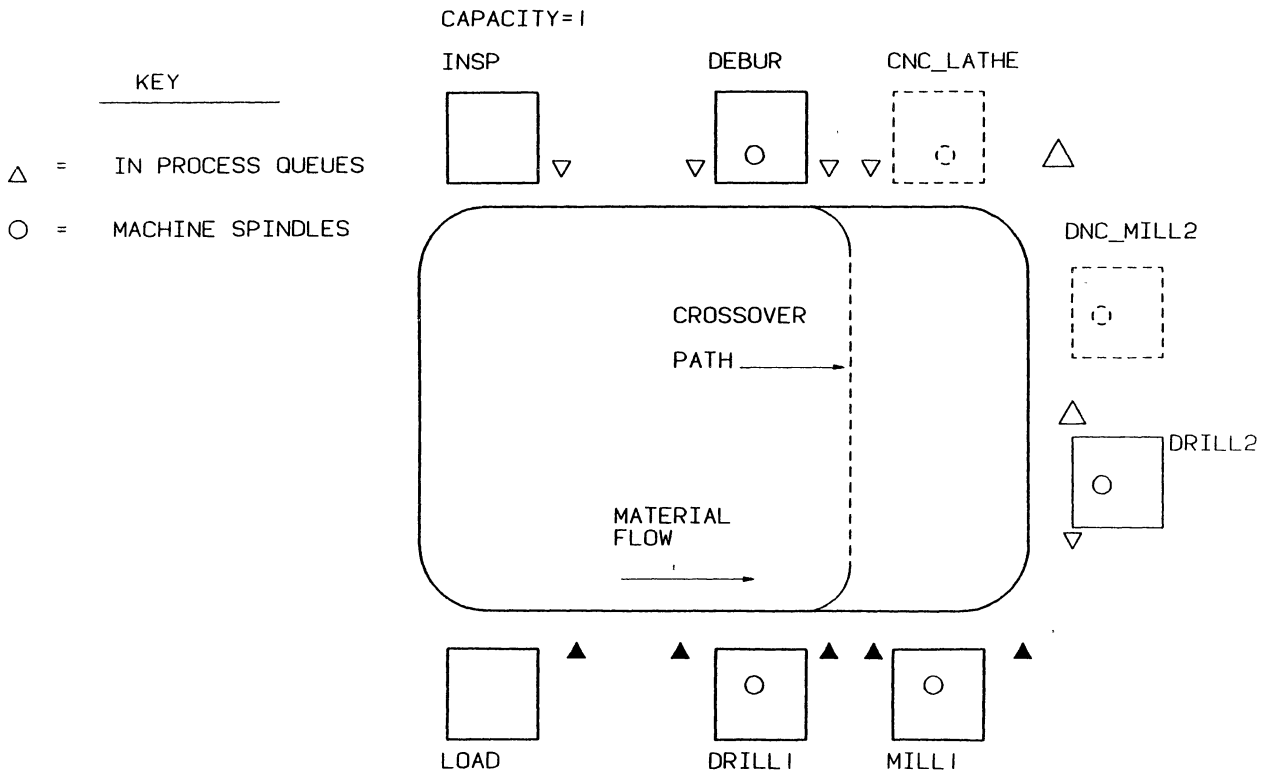


Figure 15. Example System II

Model IV. The fourth model will represent a change in the AGV system that allows for crossover paths in the loop. Also, instead of using parts in batchs, the production schedule will be modified to load one part at a time from each group. This feature illustrates how a variety of parts can be in process in the system at one time. The last change for the fourth model will be to add a three station call to the system where some parts will require processing. This feature will illustrate the ability to model machinery cells linked by an AGV system. The input and output screens are in Sections I and II of Appendix H.

Summary

This chapter has presented the error checking and execution logic along with the output capabilities of **FMS**. Also, **FMS** was compared to an established language to verify that logical results can be achieved from **FMS** models. The last item in the chapter presented four example models of two flexible manufacturing systems.

The error checking logic of **FMS** proceeds through each major module to verify that the input data is logically correct. The error check verifies that each model element is correctly referenced. Part descriptions are checked for invalid routings or work centers. The final check is to form the modules into an executable network. In the error checking stage there are two types of errors: fatal and non-fatal. Fatal errors prevent any attempt at executing

the model. Non-fatal errors simply generate messages. The execution segment takes the input network and simulates the system according to the model execution parameters.

The output capabilities of **FMS** include six different reports. Three of the reports provide information on the utilization of the system components. The fourth report provides information on production and work in process. Statistics on the size of in process queues is provided in report five. Finally, report six provides statistics on breakdowns.

The language verification section compared the performance of **FMS** and SLAM in modeling three queueing systems. This comparison showed that **FMS** can easily produce reasonable results for simple systems. The last section in the chapter demonstrated several features of **FMS** by modeling two different versions of a flexible manufacturing system.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Research Problem Restated

The primary goal of this research was to develop an interactive simulation language that would address flexible manufacturing systems. The proposed language would not require the user to posses a large amount of knowledge. Also, the language should be production oriented in reporting statistics and terminology. The types of flexible manufacturing systems that were to be addressed were those containing a mixture of robots and transfer mechanisms moving parts between work centers. Also, the flexible manufacturing systems may contain a main material movement system such as an AGV system. The language however, should not address the details between 5-axis robots and 6-axis robots. Internally the language should utilize efficient and effective data manipulation techniques.

Research Summary

The result of this research was the simulation language Flexible Manufacturing Simulator or **FMS.** The language allows the modeler to describe the elements in a flexible manufacturing simulator in terms of the delay experience by

a part moving through the system. Model elements such as work centers were assigned user defined names to aid in model communication. Work centers are described according to the number of parallel spindles and have associated with them input and output shuttles and queues. The work centers are joined in the system by shuttles or the material handling system. To add validity to a model the user may specify breakdowns and repairs to one or more work centers.

The reporting facilities in **FMS** allow for both interactive report reviewing and hard copy. Within **FMS** four types of reports are developed: utilization reports, production reports, queue reports, and breakdown reports. Included in the utilization reports are utilizations for work centers, shuttles, and the material handling system.

The internal data manipulation techniques employed by **FMS** are superior to the conventional techniques employed by existing languages. The efficiency of event calendar manipulations in **FMS** outperforms the conventional double linked list by a factor of 2^x, where x is the performance of the **FMS** algorithm. This performance is proportional $\log_2(YN)$, where N is the average calendar size and Y is the average number of events in the simulation. The source listing for **FMS** is presented in Appendix D.

Suggested Further Research

There are three main areas of research that would improve the current research. The first area to continue

would be developing better reporting capabilities. This area would involve developing new measures of performance along with expanding the statistics currently available. Also included in this area would be the possibility of graphical results and/or trace facilities. The second area of extended research would include letter editing and input capabilities. A feature that would enhance model formulation would be graphically building the model using the current set of statements. This feature would present the graphical version of the model as it is being created. The third and final area would be to include advanced logic modeling. This feature would develop complex scheduling policies and possibly emulating scheduler/controller logic.

A SELECTED BIBLIOGRAPHY

1. Baase, Sara. Computer Algorithms: Introduction to Design and Analysis. Reading, Massachusetts: Addison-Wesley Publishing, 1978.

2. Burgham, Patrick M. "Flexible Fabrication Moves in at Hughes Aircraft." Manufacturing Engineering, September, 1983, pp. 56-57.

3. Dudewicz, E. J. and T. G. Ralley. The Handbook of Random Number Generation and Testing with TESTRAND Computer Code. Columbus, Ohio: American Science Press, Inc., 1981.

4. ElMaraghy, H. A. "Simulation and Graphical Animation of Advanced Manufacturing Systems." (Unpublished paper presented at the Society of Manufacturing engineers meeting detroit, Michigan, March, 1983). Hamilton, Ontario, Canada: McMaster University, Department of Mechanical Engineering, 1983.

5. Emshoff, James R. and R. L. Sission. Design and Use of Computer Simulation Models. London, England: The MacMillan Company, 1970.

6. "FMS." Manufacturing Engineering, September, 1983, p. 49.

7. "Flexible Manufacturing Systems--Their tremendous "potential." Modern Materials Handling. 37, (September 7, 1982), pp. 52-73.

8. Franta, W. R. The Process View of Simulation. North Holland: 1977.

9. GEMS User's Manual.

10. Gordon, G. System Simulation. Englewood, New Jersey: Prentice-Hall, Inc., 1969.

11. Graybeal, W. J. and Udo W. Pooch. Simulation: Principles and Methods. Cambridge, Massachusetts: Winthrop Publishers, Inc., 1980.

12. Groover, M. P. Automation, Production Systems, and Computer Aided Manufacturing. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1980.

13. Hegland, Donald E. "Your Balance Between Adaptability." Production Engineering, 28, May, 1981, pp. 29-43.

14. Hegland, Donald E. "Flexible Manufacturing." Production Engineering, September, 1982, pp. 41-46.

15. Horizon Software Incorporated. A User's Guide to SPEED.

16. Klahorst, H. Thomas. "How to Plan Your FMS." Manufacturing Engineering, September, 1983, pp. 52-54.

17. Kleine, Harvey. "A Second Survey of User's Views of Discrete Digital Simulation." Simulation, Vol. 17, No. 2 (August, 1971).

18. Koren, Yoram. Computer Control of Manufacturing Systems. New York: McGraw-Hill Book Company, 1982.

19. Law, A. M. and W. D. Kelton. Simulation Modeling and Analysis. New York: McGraw-Hill Book Company, 1982.

20. McCollom, N. N. Personal Interview. General Dynamics Fort Worth Division, Fort Worth, Texas, November 19, 1983.

21. Mize, Joe H. Telephone Interview. Arizona State University, Tempe, Arizona, November 10, 1983.

22. Pegden, C. Dennis. Introduction to SIMAN. Calder, Pennsylvania: Systems Modeling Corp., 1982.

23. Pritsker, A. A. B. and C. D. Pegden. Introduction to Simulation and SLAM. New York: John Wiley and Sons, 1979.

24. Shannon, R. E. "Discrete Simulation User's Survey Revisited." Simulation, Vol. 19, No. 4, (May, 1973).

25. Shannon, R. E. System Simulation, the Art and the Science. Englewood Cliffs, New Jersey: Prentice Hall, 1975.

26. Tepsic, Rudy M. "How to Justify Your FMS . . . a Solid, Proven Approach for FMS Justification." Manufacturing Engineering, September, 1983, pp. 50-52.

APPENDIXES

APPENDIX A

TEST MODEL I REPORTS

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.75	0.184	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.78	0.169	0.00	0.000	0.00	0.000

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.76	0.181	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.74	0.192	0.00	0.000	0.00	0.000

Work in Process and Production Report

Work in Process

Average number of parts in the system = 4.81

Variance = 10.35

System Production

Average Time in the System = 3.042

Variance = 3.330

Part Production per unit time = 0.328

Work in Process and Production Report

Work in Process

Average number of parts in the system = 7.04

Variance = 43.84

System Production

Average Time in the System = 4.793

Variance = 15.517

Part Production per unit time = 0.208

Work in Process and Production Report

Work in Process

Average number of parts in the system = 8.11

Variance = 25.18

System Production

Average Time in the System = 5.075

Variance = 11.407

Part Production per unit time = 0.197

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	1.87	5.695	0	12
2	1.44	3.256	0	10

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	2.71	10.634	0	14
2	3.80	13.792	0	16

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	3.20	21.031	0	19
2	2.30	9.991	0	15

Hit a carriage return to continue.

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.2342E+01	0.1384E+01	0.5910E+00	0.4660E-01	0.7117E+01	560

STATISTICS FOR TIME-PERSISTENT VARIABLES

	MEAN VALUE	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	TIME INTERVAL	CURRENT VALUE
NUMBERINSYS	0.3303E+01	0.2370E+01	0.0000E+00	0.1100E+02	0.4000E+03	0.5000E+01

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	1.0540	1.5312	9	0	0.7462
2	QUEUE	0.9471	1.5268	10	3	0.6717
3	CALENDAR	2.3019	0.6435	4	3	0.3825

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.6749	0.4684	1	0.0000	3.5067	18.5735	564
2	QUEUE	1	0.6272	0.4836	1	0.0000	3.4187	15.4627	560

*** END OF DATA SET ***

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.4465E+01	0.2600E+01	0.5824E+00	0.1063E+00	0.1180E+02	647

-->

STATISTICS FOR TIME-PERSISTENT VARIABLES

	MEAN VALUE	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	TIME INTERVAL	CURRENT VALUE
NUMBERINSYS	0.7225E+01	0.5021E+01	0.0000E+00	0.2400E+02	0.4000E+03	0.2000E+01

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	2.6118	2.9794	13	0	1.6097
2	QUEUE	3.0253	3.2955	16	1	1.8646
3	CALENDAR	2.5877	0.6040	4	2	0.3479

-->

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.7767	0.4165	0	0.0000	4.7424	53.6841	649
2	QUEUE	1	0.8113	0.3913	1	0.0000	4.3609	112.1894	647

++++ BOTTOM OF DATA SET +++
-->

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY U. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.4036E+01	0.2724E+01	0.6751E+00	0.4739E-01	0.1231E+02	633

...)

STATISTICS FOR TIME-PERSISTENT VARIABLES

	MEAN VALUE	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	TIME INTERVAL	CURRENT VALUE
NUMBERINSYS	0.6388E+01	0.4544E+01	0.0000E+00	0.2300E+02	0.4000E+03	0.1000E+01

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	1.9170	2.3026	11	0	1.2095
2	QUEUE	2.8788	3.4571	15	0	1.8191
3	CALENDAR	2.5919	0.5512	4	2	0.3611

...)

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.7838	0.4116	1	0.0000	2.1271	35.7054	633
2	QUEUE	1	0.8083	0.3936	0	0.0000	2.2366	46.1987	633

++++ BOTTOM OF DATA SET +++
...)

APPENDIX B

TEST MODEL II REPORTS

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
2.24	0.964	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
2.29	0.948	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 3.63

Variance = 6.62

System Production

Average Time in the System = 1.542

Variance = 1.399

Part Production per unit time = 0.648

Work in Process and Production Report

Work in Process

Average number of parts in the system = 4.66

Variance = 15.48

System Production

Average Time in the System = 2.016

Variance = 2.826

Part Production per unit time = 0.495

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	1.34	3.786	0	9
2	0.00	0.000	0	0

Hit a carriage return to continue.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 3.29

Variance = 5.34

System Production

Average Time in the System = 1.425

Variance = 1.258

Part Production per unit time = 0.696

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	1.16	3.463	0	10
2	0.00	0.000	0	0

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	2.24	12.035	0	20
2	0.00	0.000	0	0

Hit a carriage return to continue.

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.3600E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.1575E+01	0.1319E+01	0.8379E+00	0.2441E-03	0.8888E+01	770

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	1.1609	1.8383	9	0	0.5414
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	3.2094	1.0057	5	3	0.5907

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	3	2.2095	1.0052	2	0.0000	3.0000	3.0000	770

++++ BOTTOM OF DATA SET +++

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.3600E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.1572E+01	0.1281E+01	0.8148E+00	0.1678E-02	0.7702E+01	866

-->

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	1.5139	2.3557	13	8	0.6214
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	3.3121	0.9246	5	4	0.5330

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	3	2.3122	0.9240	3	0.0000	3.0000	3.0000	866

++++ BOTTOM OF DATA SET +++
-->

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.3600E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.2108E+01	0.1773E+01	0.8409E+00	0.7431E-02	0.1049E+02	786

==>

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	2.3768	3.4984	16	2	1.0817
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	3.2531	0.9825	5	4	0.5758

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	3	2.2532	0.9820	3	0.0000	3.0000	3.0000	786

++++ BOTTOM OF DATA SET +++
-->

APPENDIX C

TEST MODEL III REPORTS

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.61	0.237	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Work Center Utilization Report

Work Center : WC1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : WC2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.68	0.216	0.00	0.000	0.00	0.000

Work Center : WC3

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	0.86	1.961	0	8
2	0.00	0.000	0	0

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	1.04	2.800	0	10
2	0.00	0.000	0	0

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	1.05	2.258	0	6
2	0.00	0.000	0	0

Hit a carriage return to continue.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 1.84

Variance = 4.42

System Production

Average Time in the System = 2.153

Variance = 4.986

Part Production per unit time = 0.464

Work in Process and Production Report

Work in Process

Average number of parts in the system = 1.47

Variance = 2.86

System Production

Average Time in the System = 1.919

Variance = 2.687

Part Production per unit time = 0.520

Work in Process and Production Report

Work in Process

Average number of parts in the system = 1.73

Variance = 3.14

System Production

Average Time in the System = 2.016

Variance = 2.693

Part Production per unit time = 0.495

SLAM SUMMARY REPORT

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.2513E+01	0.2550E+01	0.1015E+01	0.2734E-01	0.1081E+02	333

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	1.4261	2.4545	12	1	1.7028
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	1.6669	0.4714	3	2	0.7568

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.6669	0.4713	1	0.0000	5.1078	38.3855	333

++++ BOTTOM OF DATA SET +++

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY W. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.1555E+01	0.1295E+01	0.8328E+00	0.4959E-02	0.5901E+01	312

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	0.6660	1.2027	6	3	0.8430
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	1.5639	0.4960	3	2	0.7704

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.5640	0.4959	1	0.0000	8.1372	26.7368	312

++++ BOTTOM OF DATA SET +++

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT SIMULATION3703 BY U. H. REMY

DATE 11/23/1983 RUN NUMBER 1 OF 1

CURRENT TIME 0.4000E+03
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN SYSTEM	0.1676E+01	0.1407E+01	0.8395E+00	0.1758E-01	0.6439E+01	351

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	QUEUE	0.8348	1.3935	8	1	0.9460
2		0.0000	0.0000	0	0	0.0000
3	CALENDAR	1.6433	0.4791	3	2	0.7008

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	QUEUE	1	0.6434	0.4790	1	0.0000	6.7681	15.6693	351

++++ BOTTOM OF DATA SET +++
**>

APPENDIX D

FMS SOURCE LISTING

```
                FMSS : proc options(main);
     1          /*************************************************************/
     1          /*                                                           */
     1          /*              Main Structure Declarations                  */
     1          /*                                                           */
     1          /*************************************************************/
     1
     1
     1          dcl   1   Machine   (25)    external,
     1
     1                    2  M_name          char ( 30 ) var,
     1                    2  Type            char ( 10 ) var,
     1                    2  Capacity        fixed bin( 15 ),
     1                    2  Util            fixed bin( 15 ),
     1                    2  Down            fixed bin(  7 ),
     1                    2  Block           fixed bin(  7 ),
     1                    2  Sums            float bin( 31 ),
     1                    2  Sumsq           float bin( 31 ),
     1                    2  Tlast           float bin( 31 ),
     1                    2  Bypart (25) ,
     1
     1                       3  P_name          char ( 30 ) var,
     1                       3  Num             fixed bin( 15 ),
     1                       3  Last            float bin( 31 ),
     1                       3  Sumval          float bin( 31 ),
     1                       3  Sum2            float bin( 31 ),
     1                       3  CHARAC ( 15 )   float bin( 31 );
     1
     1
     1          DCL    works fixed bin(7) external;
     1
     1
     1
     1          dcl   1  Transport    ext,
     1
     1                   2  Zones ( 100 ),
     1
     1                      3  Start         char(30) var,
     1                      3  Length        float bin(31),
     1                      3  Rate          fixed bin(31),
     1                      3  D_name        char(15) var,
     1                      3  End           char(30) var,
     1                      3  Parts ( 50) ,
     1
     1                         4  P_name          char(30) var,
     1                         4  CHARAC(15)      float bin(31),
     1
     1                      3  Util            fixed bin(31),
     1                      3  last_U          fixed bin(31),
     1                      3  Sum_U         float bin(31),
     1                      3  Name2         float bin(31);
```

```
 56    1
 57    1      dcl   PATHS           fixed bin(7)  external;
 58    1
 59    1
 60    1
 61    1      /*****************************************************************/
 62    1      /*                                                               */
 63    1      /*                  Local Variable Declarations                  */
 64    1      /*                                                               */
 65    1      /*****************************************************************/
 66    1
 67    1
 68    1      dcl
 69    1           option     char(80) var;
 70    1
 71    1      dcl  err        bit(1) init('1'B);
 72    1
 73    1      dcl  true       bit(1) init('1'B);
 74    1
 75    1
 76    1
 77    1      /*****************************************************************/
 78    1      /*                                                               */
 79    1      /*                Entry  Declarations                            */
 80    1      /*                                                               */
 81    1      /*****************************************************************/
 82    1
 83    1      dcl   First_Page  entry;
 84    1
 85    1      %include 'PAUSE.PLI';
 94    1
 95    1      dcl   MCNTL     entry;
 96    1
 97    1      dcl   MODEL     entry;
 98    1
 99    1      dcl   EXEC      entry;
100    1
101    1      dcl   OUTPUT    entry;
102    1
103    1      dcl   EDIT_CNTL entry;
104    1
105    1      dcl   EDIT_MODEL  entry;
106    1
107    1      dcl   MAIN_MENU   entry;
108    1
109    1
110    1      dcl  ERRCHK   entry (float bin(31),float bin(31),bit(1));
111    1
112    1
113    1      /*****************************************************************/
114    1      /*                                                               */
115    1      /*                Main Procedure Execution                     */
116    1
117    1
118    1      call First_Page;
119    1      call Pause;
```

```
  121     1       call Pause;
  122     1
  123     1
  124     1
  125     1
  126     1           Do while(option ^= '7' );
  127     2
  128     2       Call MAIN_MENU;
  129     2
  130     2           Put skip list('Enter the option corresponding to the function you wish to execute : ');
  131     2          get edit(option)(A(80));
  132     2
  133     2
  134     2
  135     2               If (option = '1' ) then call MCNTL;
  136     2
  137     2                else
  138     2
  139     2                   If ( option = '2' )then call MODEL;
  140     2
  141     2                    else
  142     2
  143     2                      If ( option = '3' ) then call EXEC;
  144     2
  145     2                       else
  146     2
  147     2                         If(option = '4' ) then call OUTPUT;
  148     2
  149     2                          else
  150     2
  151     2                            If(option = '5' ) then call EDIT_CNTL;
  152     2
  153     2                             else
  154     2
  155     2                        if( option  = '6' ) then call EDIT_MODEL;
  156     2
  157     2
  158     2
  159     2
  160     2         END;
  161     1
  162     1       END FMSS;
  163             MAIN_MENU : proc;
  164     1
  165     1       put skip(25);
  166     1       put skip EDIT('*********************************************')(X(15),A);
  167     1       put skip EDIT('*                                           *')(X(15),A);
  168     1       put skip EDIT('*       FLEXIBLE MANUFACTURING SIMULATOR    *')(X(15),A);
  169     1       put skip EDIT('*                                           *')(X(15),A);
  170     1       put skip EDIT('*                  Main  Menu               *')(X(15),A);
  171     1       put skip EDIT('*                                           *')(X(15),A);
  172     1       put skip EDIT('*    Option          Function               *')(X(15),A);
  173     1       put skip EDIT('*                                           *')(X(15),A);
  174     1       put skip EDIT('*      1.      Enter model control parameters*')(X(15),A);
  175     1       put skip EDIT('*                                           *')(X(15),A);
  176     1       put skip EDIT('*      2.      Enter model statements       *')(X(15),A);
```

```
178     1       put skip EDIT('*      3.      Execute  model statements       *')(X(15),A);
179     1       put skip EDIT('*                                              *')(X(15),A);
180     1       put skip EDIT('*      4.      Review  output reports          *')(X(15),A);
181     1       put skip EDIT('*                                              *')(X(15),A);
182     1       put skip EDIT('*      5.      Edit control parameters         *')(X(15),A);
183     1       put skip EDIT('*                                              *')(X(15),A);
184     1       put skip EDIT('*      6.      Edit model statements           *')(X(15),A);
185     1       put skip EDIT('*                                              *')(X(15),A);
186     1       put skip EDIT('*      7.      Terminate the program           *')(X(15),A);
187     1       put skip EDIT('*                                              *')(X(15),A);
188     1       put skip EDIT('************************************************')(X(15),A);
189     1
190     1       end MAIN_MENU;
191             First_Page  : proc;
192     1       %include 'PAUSE.PLI';
201     1
202     1
203     1       put skip(25);
204     1       put skip(6);
205     1       put skip edit('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')(X(14),A);
206     1       PUT SKIP EDIT('|                                                                      |')(X(14),A);
207     1       PUT SKIP EDIT('|          FLEXIBLE MANUFACTURING SIMULATOR                            |')(X(14),A);
208     1       PUT SKIP EDIT('|                                                                      |')(X(14),A);
209     1       PUT SKIP edit('|                    Copyright 1984                                    |')(X(14),A);
210     1       put skip edit('|                          by                                          |')(X(14),A);
211     1       put skip edit('|                    William H. Remy                                   |')(X(14),A);
212     1       put skip edit('|                                                                      |')(X(14),A);
213     1       put skip edit('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')(X(14),A);
214     1
215     1
216     1       put skip(7);
217     1
218     1       call pause;
219     1       call pause;
220     1
221     1       end First_Page;
222
223             ERRCHK : PROC(a,b,e);
224     1
225     1       /*  Procadure to check it 1) a or b are < 0
226     1                                 2) a > b
227     1       */
228     1
229     1       dcl e bit(1);
230     1
231     1       dcl (a,b) float bin(31);
232     1
233     1       If(a < 0.)then do;
234     2                       put skip(3) list('Invalid entry=> parm1 must be non-negative');
235     2                       e = '1'b;
236     2                       end;
237     1           else
238     1
239     1       If(b< 0.)then do;
240     2                       put skip(3) list('Invalid entry=> parm2 must be non-negative');
241     2                       e = '1'b;
                                return;
```

```
243  2                        end;
244  1
245  1        else
246  1           If(b <= a)then do;
247  2                           put skip(3) list('Invalid sequence => parm2 > parm1');
248  2                           e = '1'B;
249  2                           end;
250  1
251  1
252  1          else
253  1                 e = '0'B;
254  1
255  1        end ERRCHK;
256           ERROR : PROC( enum, s );
257  1
258  1        dcl enum     fixed bin(15);
259  1
260  1        dcl s        char(80) var;
261  1
262  1        if( enum = 140 ) then do;
263  2
264  2                  put skip(2) list('===> Error  # 140 ');
265  2                  put skip    list('Your Work Center descriptons do not contain');
266  2                  put skip    list('a load station or an unload station.');
267  2                  end;
268  1
269  1        else  if(enum = 141 ) than do;
270  2
271  2                  put skip(2) list('===> Error  # 141');
272  2                  put skip    list('The Work Center name',s);
273  2                  put skip    list('does not exist. Try Again !');
274  2                  end;
275  1
276  1        else  if(enum = 110 ) then do;
277  2
278  2                  put skip(2) list('===> Error  # 110');
279  2                  put skip    list('The input parameter must be greater than zero.');
280  2                  end;
281  1
282  1        else  if(enum = 111 ) then do;
283  2
284  2                  put skip(2) list('===> Error # 111');
285  2                  put skip    list('The input parameter must be numeric.');
286  2                  end;
287  1
288  1        else  if(enum = 101) then do;
289  2
290  2                  put skip(2) list('===> Error  # 101');
291  2                  put skip    list('The parameter entered is not numeric.');
292  2                  end;
293  1
294  1
295  1        else  if ( enum = 201 ) then do;
296  2
297  2                    put skip(2) list('===>  ERROR  #201');
298  2                    put skip    list('The number of queues exceeds 100.');
299  2                    put skip    list('The last set of parameters was associated to queue 100');
```

```
300  2                  put skip    list("Reformulation of the modal may be necessary");
301  2                  end;
302  1
303  1          else if( enum = 130) then do;
304  2
305  2                     put skip(2) list("===>  ERROR  #130");
306  2                     put skip list("The mean must be greater than zero! ");
307  2                     end;
308  1
309  1
310  1
311  1          else if(enum = 131 )then do;
312  2
313  2                    put skip(2) list("===>  ERROR  #131");
314  2                    put skip edit(s," is an invalid dispatching rule.")(A,A);
315  2                    end;
316  1
317  1          else  put skip list("DBG ERROR",enum,s);
318  1
319  1          end ERROR;
320
321
322             LOGIC  :  proc;
323  1          dcl  error1  bit(1),
324  1               true   bit(1) init('1'B),
325  1               false  bit(1) init('0'B),
326  1               load   bit(1),
327  1               unload bit(1);
328  1
329  1
330  1          dcl  count fixed bin(7),
331  1               JJ    fixed bin(7);
332  1
333  1          dcl  string  char (80) var;
334  1          dcl  ERROR   entry(fixed bin(15),char(80) var);
335  1          % INCLUDE  "UCASE.PLI";
336  1
346  1
347  1
348  1          %INCLUDE "MACHINE.PLI";
349  1
373  1
374  1
375  1
376  1
377  1
378  1          /*********************************************************/
379  1
380  1
381  1
382  1          load = false;   unload = false;
383  1
384  1          do JJ = 1 to works by 1;
385  2             if( Type ( JJ ) = "L" ) then load = true;
386  2
387  2
```

```
  389     2
  390     2         end;
  391     1
  392     1
  393     1       do while ( ^load  | ^ unload  );
  394     2
  395     2
  396     2          Call ERROR(140,'  ');
  397     2
  398     2            error1 = true;
  399     2
  400     2              do while (  ^ load );
  401     3
  402     3                count = 0;
  403     3
  404     3                put skip(2) list('Enter tha work Center name you would');
  405     3                put skip    list('like to make a "load" station : ');
  406     3
  407     3                get list(string);
  408     3                string = UCASE(string);
  409     3
  410     3                 do while ( error1 &   count < works );
  411     4
  412     4
  413     4
  414     4                 count = count + 1 ;
  415     4                 if( M_name( count ) = string ) then error1 = false;
  416     4
  417     4
  418     4                  end;
  419     3
  420     3                 if(error1) then call ERROR (141,string);
  421     3
  422     3
  423     3                 Type(count ) = 'L';
  424     3                 load = true;
  425     3
  426     3
  427     3                end;
  428     2
  429     2
  430     2
  431     2       do while ( ^unload );
  432     3
  433     3             error1 = true;
  434     3
  435     3             do while ( ^unload );
  436     4
  437     4               count = 0;
  438     4               put skip(2) list('Enter tha Work Center name you wish to ');
  439     4               put skip    list('make an "unload" station : ');
  440     4               get list(string);
  441     4
  442     4               string = UCASE(string);
  443     4
  444     4                do while ( error1 & count < works );
```

```
                    count = count + 1 ;
                     if(M_name(count) = string) then error1 = false;

                     end;

                if( error1 ) then call ERROR (141,string);

               end;

                Type(count ) = 'UL';
                unload = true;

             end;

      end;

      End  LOGIC;
      lex2 : proc(s,val,err);

      dcl s   char(80) var;

      dcl val float bin(31);

      dcl err bit(1);

      dcl digits char(15) var init('0123456789');

      dcl (front,end,j,position) fixed bin(7);

      dcl symbol char(1) var;

      position = 1;

      end = LENGTH(s);

      j = 1;
      err = '0'B;

      do while ( ^err   & j <= end);


         symbol = SUBSTR(s,j,1);
         position = INDEX(digits,symbol);


         if( position = 0  &  symbol ^= '.')then err = '1'B;
         j = j + 1;

         end;

      if( err ) then val = 0.;

      else
      if( j > end ) then val = SUBSTR(s,1);
```

```
  503     1
  504     1       end lex2;
  505
  506             MODEL : proc ;
  507     1
  508     1       /*  This procedure is for inputting the model statements
  509     1           It will call the appropriate procedures to input the statements
  510     1           for  machines, shuttles, the primary material handling system,
  511     1           and for break downs.
  512     1
  513     1       */
  514     1
  515     1       dcl
  516     1
  517     1            DOWN     entry,
  518     1
  519     1            MACH     entry,
  520     1
  521     1             TRANS    entry,
  522     1
  523     1            SHUT     entry;
  524     1
  525     1       dcl ans  char(1);
  526     1
  527     1
  528     1        dcl error bit(1) init('0'B);
  529     1
  530     1       /**************    Procedure execution   segment  **************/
  531     1
  532     1       Call   menu;
  533     1
  534     1
  535     1        error = '1'B;
  536     1
  537     1        do while(error);
  538     2
  539     2       Put skip list('Chose an option by entering the appropriate number.');
  540     2       Put skip list('Which option ? ');
  541     2       get list(ans) ;
  542     2
  543     2       Call check;
  544     2          end;
  545     1
  546     1
  547     1
  548     1            Do while ( ans ^= '5');
  549     2
  550     2              If ( ans = '1' ) then call MACH;
  551     2
  552     2              else
  553     2                  If ( ans = '2' ) then call SHUT;
  554     2
  555     2                 else
  556     2                     If ( ans = '3' ) then call TRANS;
  557     2
  558     2                        else
```

```
                    else
                       If( ans ^= '5' )then call check;


                     Call menu;
                     Put skip list('Chose an option by entering the appropriate number.');
                     Put skip list('Which option ? ');
                     get list(ans) ;


          end;  /* end of input sequence  */

     /* !!!!!!!!!!!!    Internal   Procadures    !!!!!!!!!!!!!!!! */

     menu : proc;

     put skip (25);
     put skip list('\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/');
     put skip list('\                                                                    /');
     put skip list('\  Option              Module Description                            /');
     put skip list('\                                                                    /');
     put skip list('\     1.            Enter the Work Center descriptions             /');
     put skip list('\     2.            Enter the Shuttle descriptions                 /');
     put skip list('\     3.            Enter the Material Handling description        /');
     put skip list('\     4.            Enter the Breakdown descriptions               /');
     put skip list('\     5.            Return                                         /');
     put skip list('\                                                                    /');
     put skip list('\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/');


     end menu;


     check : proc;

     DCL string char(6) var;

     dcl position fixed bin(7);

     string = '12345';

     position = INDEX(string,ans);

      If ( position = 0 )then do;

                    Put skip list('Invalid  option, Re-enter');
               error = '1'B;

              end;


       else  error = '0'B;
```

```
617  1
618  1
619  1     end MODEL;
620
621
622           GPT : PROC;
623  1
624  1     DCL NULL  char(80) var  init("");
625  1
626  1        dcl(  Pname ,           /* Project Name  */
627  1              Mname )           /* Modelers Name */ char(80) var ext;
628  1
629  1
630  1        dcl date1  char(80) var ext; /*  Current date1   */
631  1
632  1     Pname = NULL;Mname = NULL;date1 = NULL;
633  1
634  1
635  1
636  1     do while (Pname = NULL);
637  2
638  2                  PUT SKIP(25);
639  2                  PUT SKIP LIST('Enter a project name less then 80 characters');
640  2                  PUT SKIP LIST('Project name = ');
641  2                    get edit (Pname) (A(80));
642  2
643  2                   END;
644  1
645  1     do while (Mname = NULL );
646  2
647  2
648  2                put skip(5);
649  2                put skip list('Enter a modelers name with less than 80 characters.');
650  2                   put skip list('Modelers name = ');
651  2                       get edit(Mname) ( A(80));
652  2
653  2
654  2                    end;
655  1
656  1      do while ( date1 = NULL );
657  2
658  2
659  2               put skip(5);
660  2                        Put skip list('Enter the current date ');
661  2                   put skip list('data  = ');
662  2                  get edit(date1)  (A(80));
663  2
664  2
665  2                       end;
666  1
667  1
668  1        End GPT; /* End of Project tittles input */
669           /*      This procedure is used to enter the information about the  control parameters for the execution
670              There are four menus which allow the user to select any module
671              of input.  Control of the program is then passed to procedures
672              to input information about the actual model itself.
```

```
674
675     1          */
676
677
678                    Mcntl : proc:
679     1
680     1
681     1              dcl
682     1                ( Pname  ,          /*  Project name  */
683     1                  date1  ,
684     1                  Mname  )          /*  Modelers name */
685     1
686     1                              Char (80) var  ext;
687     1
688     1
689     1            dcl
690     1                 (tstart        ,
691     1                 tfinish        ,
692     1                 sclear         ,
693     1                 strace         ,
694     1                 ftrace         )        float bin(31)   ext;
695     1
696     1           dcl partmax                   fixed bin(15)   ext;
697     1
698     1                  dcl arr bit(1) init('1'B);
699     1
700     1
701     1
702     1                             Dcl
703     1                               (option0,option1,option2,option3) fixed bin(7) init(0  );
704     1
705     1
706     1             dcl   ERRCHK  entry (float bin(31),float bin(31),bit(1));
707     1
708     1            dcl  Partsys  entry;
709     1
710     1            dcl  Stats    entry;
711     1
712     1             dcl  Header  entry;
713     1
714     1
715     1            /*****************    MAIN PROCEDURE EXECUTION  ******************/
716     1
717     1             err = '1'B;
718     1
719     1
720     1                       Call Menu0; /* print the first menu and chack input option  */
721     1              do while ( err );
722     2
723     2
724     2                       put skip list('Please enter your option: which option ?');
725     2                       get list(option0);
726     2
727     2
728     2                      Call ERRCHK(option0,5,err);
729     2
730     2
```

```
                end; /* end of option input loop  */

      Do while (option0 < 4);

                If (option0 = 1)then call Header;

                  else If (option0 = 2) then call Partsys;

                         else  call Stats;


     err = '1'B;

                Call Menu0; /* print the first menu and check input option  */
     do while ( err );

                put skip list('Please enter your option: which option ?');
                get list(option0);

               Call ERRCHK(option0,5,err);

                end; /* end of option input loop  */
                       End;  /* End of parameter input  */


 /*********************************************************************/
/*                INTERNAL PROCEDURES                                */
/*********************************************************************/

Menu0  : proc;

put skip(25);
put skip list('*********************************************************************');
put skip list('*                                                                   *');
put skip list('*           MODEL CONTROL  INFORMATION                              *');
put skip list('*                                                                   *');
put skip list('*     Option                    Parameters                          *');
put skip list('*                                                                   *');
put skip list('*       1.             Header and Project Information              *');
put skip list('*       2.             Part  Information                           *');
put skip list('*       3.             Statistics Gathering Information            *');
put skip list('*       4.             Return                                      *');
put skip list('*                                                                   *');
put skip list('*********************************************************************');

        End Menu0;

/*****************************************************************************/
```

```
  788
  789             Header : proc;
  790     1
  791     1
  792     1       DCL GPT ENTRY;
  793     1
  794     1       DCL MEP ENTRY;
  795     1
  796     1       DCL ERRCHK  ENTRY  (FLOAT BIN(31),FLOAT BIN(31), BIT(1));
  797     1       DCL option1  fixed bin(7);
  798     1
  799     1       dcl err bit(1) init('1'B);
  800     1
  801     1
  802     1
  803     1         option1 = 0;
  804     1
  805     1
  806     1
  807     1
  808     1        Call menuz;
  809     1
  810     1
  811     1         err = '1'B;
  812     1
  813     1          Do while ( err );
  814     2
  815     2
  816     2             put skip list('Select the desired option by entering a number,');
  817     2             put skip list(' which option ?');
  818     2             get list(option1);
  819     2
  820     2
  821     2               call ERRCHK(option1,4,err);
  822     2
  823     2              End; /* option input and error check */
  824     1
  825     1
  826     1       Do while ( option1 < 3 );
  827     2
  828     2           If(option1 = 1)then call GPT;
  829     2
  830     2
  831     2                         else call MEP;
  832     2
  833     2        Call menuz;
  834     2
  835     2
  836     2         err = '1'B;
  837     2
  838     2          Do while ( err );
  839     3
  840     3
  841     3             put skip list('Select the desired option by entering a number,');
  842     3             put skip list(' which option ?');
  843     3             get list(option1);
```

```
845     3
846     3               call ERRCHK(option1,4,err);
847     3
848     3             End; /* option input and error check */
849     2
850     2
851     2                 end; /*   end of header loop  */
852     1
853     1
854     1       menuz : proc;
855     2
856     2
857     2       put skip(25);
858     2       put skip list("#################################################################");
859     2       put skip list("#                                                               #");
860     2       put skip list("#              HEADER / PROJECT INFORMATION                     #");
861     2       put skip list("#                                                               #");
862     2       put skip list("#    Option                      Item                           #");
863     2       put skip list("#                                                               #");
864     2       put skip list("#      1.              General project tittles                  #");
865     2       put skip list("#                                                               #");
866     2       put skip list("#      2.              Model execution parameters               #");
867     2       put skip list("#                                                               #");
868     2       put skip list("#      3.              Return                                   #");
869     2       put skip list("#                                                               #");
870     2       put skip list("#################################################################");
871     2
872     2       end menuz;
873     1
874     1
875     1       End Header;
876     1       MEP : proc;
877   | 1       /*************** ENTRY DECLARATIONS ******************/
878     1
879     1         %INCLUDE "UCASE.PLI";
889     1       DCL ERRCHK ENTRY (float bin(31),float bin(31),bit(1));
890     1
891     1
892     1
893     1
894     1
895     1
896     1       dcl lex2  entry  ( char(80) var,float bin(31) , bit(1));
897     1
898     1       dcl ERROR  entry ( fixed bin(31),char(*) var);
899     1
900     1
901   | 1       /*****************  VARIABLE DECLARATIONS  ****************/
902     1
903     1
904     1        dcl( strace   ,   /* Time to start trace  */
905     1             ftrace   ,   /* Time to finish trace */
906     1           tstart,
907     1           tfinish,
908     1            sclear   )   /* Time to clear statistics  */
909     1
910     1                         Float bin(31) ext;
```

```
911     1
912     1          dcl partmax fixed bin(15) ext;
913     1       dcl  MAXCARTS      fixed bin(15) external;
914     1
915     1
916     1
917     1       dcl err bit(1) init('0'B);
918     1       dcl true bit(1) init('1'B);
919     1       dcl false bit(1) init('1'B);
920     1       dcl  value float bin(31);
921     1
922     1
923     1
924     1       dcl  enum  fixed bin(31);
925     1       dcl (string1, string2 ) char(80) var;
926     1
927     1       dcl(ans,ans1,ans2,ans3) char(3) var;
928     1
929     1
930 |   1       /************* Procedure   Execution   Segment  ***************/
931     1
932     1          err = true;
933     1
934     1
935     1          do while ( err );
936     2
937     2              put skip(25) list('Enter the starting and stopping times for ');
938     2             put skip list('the simulation seperating the values by commas.');
939     2             put skip list('Times =  ');
940     2             get list(string1,string2);
941     2
942     2           Call lex2( string1,tstart,err);
943     2
944     2            if( err ) then call ERROR(101,string1);
945     2
946     2           Call lex2(string2,tfinish,err);
947     2
948     2             if( err ) then call ERROR(101,string2);
949     2
950     2       if ( ^ err ) then
951     2
952     2
953     2                Call ERRCHK(tstart,tfinish,err);
954     2
955     2
956     2               end;
957     1
958     1
959     1       ans = '';
960     1       err = true;
961     1
962     1       do while( ( ans ^= 'YES') & (ans ^= 'NO'));
963     2
964     2       PUT SKIP LIST(ans);
965     2          put skip(5) list('Do you wish to use the TRACE option ? (Yes/No)');
966     2          get file(SYSIN)  list(ans);
```

```
   968     2          ans = UCASE(ans);
   969     2          If((ans ^= 'YES') & (ans ^= 'NO '))then put skip list('Invalid, Re-enter ?');
   970     2
   971     2          end; /* end of trace input and error check  */
   972     1
   973     1        do while(( ans = 'YES' ) & err);
   974     1
   975     2          put skip(3) list('Enter the starting and stopping times for');
   976     2          put skip list('the TRACE, seperating the values by commas.');
   977     2          put skip list('Times = ');
   978     2          get list(string1,string2);
   979     2
   980     2           Call  lex2(string1,strace,err);
   981     2
   982     2            if( err ) then call ERROR(101,string1);
   983     2
   984     2           Call  lex2(string2,ftrace,err);
   985     2
   986     2            if( err ) then call ERROR(101,string2);
   987     2
   988     2
   989     2        if ( ^ err) then
   990     2                      Call ERRCHK(strace,ftrace,err);
   991     2
   992     2
   993     2            if( err ) then call ERROR(101,' ');
   994     2               else
   995     2                    Call Validity(strace,ftrace);
   996     2
   997     2
   998     2          end; /* end of trace input with error check  */
   999     2
  1000     1        ans1 = '';
  1001     1
  1002     1        do while( ( ans1 ^= 'YES') & (ans1 ^= 'NO'));
  1003     1
  1004     2          Put skip(2) list('Do you wish to have statistics cleared');
  1005     2          put skip list('after a certain clock time ? (Yes/No)');
  1006     2          get file(SYSIN) edit(ans1) (A(3));
  1007     2
  1008     2          ans1 = UCASE(ans1);
  1009     2          If((ans1 ^= 'YES') & (ans1 ^= 'NO'))then put skip(2) list('Invalid, Re-enter?');
  1010     2
  1011     2           end;
  1012     1
  1013     1        err = true;
  1014     1
  1015     1        do while((ans1 = 'YES') & err);
  1016     1
  1017     2          put skip(3) list('Enter the time you would like to have');
  1018     2          put skip list('the statistical arrays cleared.');
  1019     2          put skip list('Time =  ');
  1020     2          get list(string1);
  1021     2
  1022     2          Call  lex2(string1,sclear,err);
  1023     2
```

```
1025    2               if( err ) then call ERROR(101,' ');
1026    2
1027    2
1028    2
1029    2       if( ^err ) then
1030    2          Call ERRCHK(sclear,1.5+6,err);
1031    2
1032    2
1033    2         if ( err ) then call ERROR( 101,' ');
1034    2
1035    2          else
1036    2                Call Validity(sclear,tstart);
1037    2                    Call Validity(tstart,sclear);
1038    2
1039    2
1040    2          end;
1041    1
1042    1
1043    1       err = true;
1044    1       do while ( err );
1045    2
1046    2
1047    2          put skip(3) list('What is the TOTAL number of parts ');
1048    2          put skip list('allowed in the system at one time?');
1049    2          put skip list('Max parts = ');
1050    2       get edit(stringl) ( A(80));
1051    2       call lex2(stringl,value,err);
1052    2       if ( value < 1.0 ) then do;
1053    3                put skip(2) list('Invalid');
1054    3                err = true;
1055    3                end;
1056    2
1057    2
1058    2          end;
1059    1
1060    1
1061    1       partmax = CEIL(value);
1062    1
1063    1
1064    1       err = true;
1065    1       do while ( err );
1066    2
1067    2             put skip(3) list('What is the Total number of CARRIERS');
1068    2             put skip list('available in the Material Handling System?');
1069    2             put skip list('Max CARRIERS = ');
1070    2             get edit( stringl ) (A(80));
1071    2             call lex2(stringl,value,err);
1072    2             if( err ) then put skip(2) list('Invalid entry');
1073    2             end;
1074    1
1075    1       MAXCARTS = CEIL ( value);
1076    1
1077    1
1078    1       /*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
1079    1
1080    1
```

```
1082    2
1083    2       dcl val1  float bin(31),
1084    2           val2  float bin(31);
1085    2
1086    2       if( val1 < tstart ) then err = '1'B;
1087    2
1088    2         else
1089    2
1090    2              if( val2 > tfinish ) then err = '1'B;
1091    2
1092    2               else
1093    2
1094    2                       err = '0'B;
1095    2
1096    2       end Validity;
1097    1
1098    1
1099    1       END MEP;
1100            Stats : proc;
1101    1
1102    1             dcl
1103    1                  1  STATS              ext,
1104    1
1105    1                     2  work_in_process    bit(1),
1106    1                     2  production         bit(1),
1107    1                     2  utilization        bit(1),
1108    1                     2  inv_queue          bit(1),
1109    1                     2  breaks             bit(1);
1110    1
1111    1             dcl  ans char (3) var;
1112    1
1113    1             dcl  err bit(1) init('1'B);
1114    1
1115    1             %include  'UCASE.PLI';
1125    1
1126    1
1127    1
1128    1        /*************************************************************/
1129    1
1130    1       work_in_process = '0'B;
1131    1       production      = '0'B;
1132    1       utilization     = '0'B;
1133    1       inv_queue       = '0'B;
1134    1
1135    1
1136    1         ans = 'NO';
1137    1         err = '1'B;
1138    1
1139    1         put skip(25);
1140    1
1141    1         put skip list('Would you like the Work in Process Summary Report ?');
1142    1
1143    1         do while ( err );
1144    2
1145    2            put skip list('Reply (Yes/No) : ');
1146    2
```

```
1148  2          ans = UCASE(ans);
1149  2
1150  2
1151  2          If(( ans ^= 'YES') & ( ans ^= 'NO' ))then do;
1152  3                                                    put skip(3) list('Invalid => Re-enter');
1153  3                                                    err = '1'B;
1154  3
1155  3                                                        end;
1156  2
1157  2
1158  2     else if( ans = 'YES' ) then do;
1159  3
1160  3                    err = '0'3;
1161  3                    work_in_process = '1'3;
1162  3                    end;
1163  2
1164  2     else if( ans = 'NO' ) then do;
1165  3                                    err = '0'B;
1166  3                                    work_in_process = '1'B;
1167  3                                    end;
1168  2
1169  2
1170  2           end;
1171  1
1172  1
1173  1          err = '1'B;
1174  1          ans = 'NO';
1175  1
1176  1          put skip(5) list('Would you like a Production Summary Report ?');
1177  1
1178  1          do while ( err );
1179  2
1180  2              put skip list( 'Reply ( Yes / No ) : ');
1181  2              get list(ans);
1182  2              ans = UCASE(ans);
1183  2
1184  2               If(( ans ^= 'YES') & ( ans ^= 'NO'))then do;
1185  3                                   put skip(2) list('Invalid => Re-enter');
1186  3                                   err = '1'B;
1187  3                                   end;
1188  2
1189  2     else if ( ans = 'YES' ) then do;
1190  3
1191  3                       production = '1'B;
1192  3                       err        = '0'3;
1193  3                       end;
1194  2
1195  2
1196  2     else if (ans = 'NO' ) then do;
1197  3                                    production = '0'B;
1198  3                                    err = '0'B;
1199  3                                    end;
1200  2
1201  2
1202  2            end;
1203  1
```

```
1205    1
1206    1          err = '1'B;
1207    1          ans = 'NO';
1208    1
1209    1           put skip(5) list('Would you like to see the Utilization Summary Report ?');
1210    1
1211    1            do while (err );
1212    2
1213    2               put skip list('Reply ( Yes/No ) : ');
1214    2               get list(ans);
1215    2               ans= UCASE(ans);
1216    2
1217    2               If(( ans ^= 'YES') & (ans ^= 'NO'))then do;
1218    3                                  put skip(3) list('Invalid => Re-enter');
1219    3                                  err = '1'B;
1220    3                                  end;
1221    2
1222    2
1223    2       else if(ans = 'YES' ) then do;
1224    3
1225    3                          utilization = '1'B;
1226    3                          err         = '0'B;
1227    3                          end;
1228    2
1229    2
1230    2       else if(ans = 'NO' ) then do;
1231    3                                    utilization = '0'B;
1232    3                                    err = '0'B;
1233    3                                    end;
1234    2
1235    2
1236    2          end;
1237    1
1238    1
1239    1        err = '1'B;
1240    1         ans = 'NO';
1241    1
1242    1         out skip(5) list('Would you like to see the Queue Summary Report ?');
1243    1
1244    1          do while ( err );
1245    2
1246    2             put skip list('Reply ( Yes/No ) : ');
1247    2             get list(ans);
1248    2             ans = UCASE(ans);
1249    2
1250    2             If(( ans ^= 'YES' ) & ( ans ^= 'NO'))then do;
1251    3                                                      put skip(2) list('Invalid => Re-enter');
1252    3                                                      err = '1'B;
1253    3                                                      end;
1254    2
1255    2
1256    2       else if( ans = 'YES') then do;
1257    3
1258    3                           inv_queue = '1'B;
1259    3                           err       = '0'B;
1260    3                           end;
```

```
1262    2
1263    2       else if( ans = 'NO')then do;
1264    3                                   inv_queue = '0'B;
1265    3                                   err =  '0'B;
1266    3                                    end;
1267    2
1268    2
1269    2           end;
1270    1
1271    1
1272    1        err = '1'B;
1273    1         ans = 'NO';
1274    1
1275    1         put skip(5) list('Do you have any Breakdowns in the model to be reported on ?');
1276    1
1277    1          do while ( err );
1278    2
1279    2             put skip list('Reply ( Yes/No ) : ');
1280    2             get list(ans);
1281    2             ans = UCASE(ans);
1282    2
1283    2             If(( ans ^= 'YES' ) & ( ans ^= 'NO'))then do;
1284    3                                                     put skip(2) list('Invalid => Re-enter');
1285    3                                                     err = '1'B;
1286    3                                                     end;
1287    2
1288    2
1289    2       else if( ans = 'YES') than do;
1290    3
1291    3                            breaks  = '1'B;
1292    3                           err      = '0'B;
1293    3                           end;
1294    2
1295    2
1296    2       else  if ( ans = 'NO' ) then do;
1297    3                                       breaks = '0'B;
1298    3                                       err = '0'B;
1299    3                                        end;
1300    2
1301    2
1302    2           end;
1303    1       END Stats;
1304             WCEDIT :  proc ;
1305    1
1306    1       %include  'UCASE.PLI';
1316    1
1317    1
1318    1       dcl   1   Machine   (25)    external,
1319    1
1320    1                 2  M_name         char ( 30 ) var,
1321    1                 2  Type           char ( 10 ) var,
1322    1                 2  Capacity       fixed bin( 15 ),
1323    1                 2  Util           fixed bin( 15 ),
1324    1                 2  Down           fixed bin(  7 ),
1325    1                 2  Block          fixed bin(  7 ),
1326    1                 2  Sums           float bin( 31 ),
```

```
1328    1                2  Tlast          float bin( 31 ),
1329    1                2  Bypart (25) ,
1330    1
1331    1                   3  P_name          char ( 30 ) var,
1332    1                   3  Nua             fixed bin( 15 ),
1333    1                   3  Last            float bin( 31 ),
1334    1                   3  Suaval          float bin( 31 ),
1335    1                   3  Sua2            float bin( 31 ),
1336    1                   3  CHARAC ( 15 )   float bin( 31 );
1337    1
1338    1
1339    1       DCL    works fixed bin(7) external;
1340    1
1341    1       dcl (string1,string2,string3,string4 ) char(80) var;
1342    1
1343    1       dcl (J,K,num ) fixed bin(7);
1344    1       dcl found  bit(1),       err     bit(1),
1345    1           true    bit(1) init('1'B),  false   bit(1) init('0'B);
1346    1
1347    1       dcl     value float bin(31);
1348    1
1349    1       dcl     lex2    entry(char(80) var,float bin(31), bit(1)),
1350    1               OPTIONS entry(char(80) var),
1351    1               SEARCH          entry(char(30) var,fixed bin(7)),
1352    1               Look    entry   (char(80) var,fixed bin(7));
1353    1
1354  | 1       /*****************************************************************************/
1355    1
1356    1       put skip(25) edit('Work Center Editor')(X(30) ,A);
1357    1       put skip(2) edit('Valid commands are: CHANGE(C), ADD(A), DELETE(D),EXIT(E)')(X(5),A);
1358    1       PUT SKIP EDIT('Command =  ')(x(5),A);
1359    1       get edit(string1)(a(80));
1360    1       string1 = UCASE(string1);
1361    1       string1 = SUBSTR(string1,1,1);
1362    1       do while (string1 ^= 'E');
1363    2
1364    2               if( string1 = 'C')then call CHNG_WC;
1365    2               else if(string1 = 'A')then call ADDWC;
1366    2               else if(string1 = 'D')then call NUKE;
1367    2               else if(string1 ^= 'E')then do;
1368    3
1369    3                       put skip edit('Invalid command.')(X(5),A);
1370    3                       put skip(2) edit('Valid commands are: CHANGE(C), ADD(A), DELETE(D),EXIT(E)')(X(5),A);
1371    3                       put SKIP EDIT('Command =  ')(x(5),A);
1372    3                       get edit(string1)(a(80));
1373    3                       string1 = UCASE(string1);
1374    3                       string1 = SUBSTR(string1,1,1);
1375    3                       end;
1376    2
1377    2               end;
1378    1
1379  | 1       /*****************************************************************************/
1380    1
1381    1       CHNG_WC :       proc;
1382    2
1383    2       put skip(3) edit('Which work center would you like to change ?  ')(X(5),A);
```

```
1385  2       string2 = UCASE(string2);
1386  2       call look(string2,num);
1387  2       put skip(2) edit('Would you like to change the NAME, TYPE, or CAPACITY ?  ')(X(2),A);
1388  2       put skip edit('Enter  NAME = N',',TYPE = T',',CAPACITY = C',',  ')(X(5),A,SKIP,X(5),A,SKIP,X(5),A,A);
1389  2       do while(string3 ^= 'N' | string3 ^= 'T' | string3 ^= 'C');
1390  3
1391  3               get edit(string3)(A(80));
1392  3               string3 = UCASE(string3);
1393  3               string3 = SUBSTR(string3,1,1);
1394  3       if( string3 ^= 'N' | string3 ^= 'C' | string3  ^= 'T')then do;
1395  4
1396  4
1397  4                       put skip edit(string3,' is not a valid option.')(A,A);
1398  4                       put skip edit('Re-enter  :  ')(A);
1399  4                       end;
1400  3
1401  3               end;
1402  2
1403  2
1404  2               if( string3 = 'N' ) then do;
1405  3
1406  3                       put skip(2) edit('Enter the new name:  ')(X(10),A);
1407  3                       get edit( string4)(A(80));
1408  3                       string4 = UCASE(string4);
1409  3                       M_name(num) = string4;
1410  3                       call SEARCH(M_name(num),works);
1411  3                       end;
1412  2
1413  2               else if( string3 = 'T' ) then do;
1414  3
1415  3                       put skip(2) edit('Enter the new type :  ')(X(10),A);
1416  3                       get edit( string4)(A(80));
1417  3                       string4 = UCASE(string4);
1418  3                       if( string4 ^= 'WC' |string4 ^= 'UL' | string4 ^= 'L') then err = true;
1419  3
1420  3                       do while ( err );
1421  4
1422  4                               put skip edit(string4 ,' is an invalid work center type.')(X(5),A);
1423  4                               put skip edit('Re-enter :  ')(X(5),A);
1424  4                               get edit( string4)(A(80));
1425  4                               string4 =UCASE(string4);
1426  4                               if( string4 ='WC' | string4 = 'UL' | string4 = 'L'  ) then err = false;
1427  4
1428  4                               end;
1429  3
1430  3                       Type(num) = string4;
1431  3                       end;
1432  2
1433  2               else if( string3 = 'C' ) then do;
1434  3
1435  3                       put skip(2) edit('Enter the new capacity :  ')(X(10),A);
1436  3                       get edit( string4)(A(80));
1437  3                       call lex2 ( string4,value,err);
1438  3                       if( value > 127.0 ) then err = true;
1439  3
1440  3                       do while (err);
```

```
1442    4                               put skip edit('The new capacity is out of range!')(X(5),A);
1443    4                               put skip edit('The capacity must be greater than zero,')(X(5),A);
1444    4                               put skip edit('and less than 128.')(X(5),A);
1445    4                               put skip edit('Re-enter : ')(X(5),A);
1446    4                               get edit( string4)(A(80));
1447    4                               call lex2(string4,value,err);
1448    4                               if( value > 127.0 ) then err = true;
1449    4                               end;
1450    3
1451    3                       Capacity(num) = CEIL(value);
1452    3                       end;
1453    2
1454    2
1455    2       end CHNG_WC;
1456    1
1457 |  1       /*****************************************************************************/
1458    1
1459    1       ADDWC : proc;
1460    2
1461    2       if ( works = 25 ) then do;
1462    3
1463    3               err = true;
1464    3               put skip(3) edit('ERROR ==>  You already have 25 work centers')(X(5),A);
1465    3               put skip edit('You cannot add more')(X(11),A);
1466    3               end;
1467    2
1468    2       do while(^err);
1469    3               works = works + 1;
1470    3               put skip(2) edit('Enter a name for the work center :  ')(X(5),A);
1471    3               get edit(string3)(A(80));
1472    3               M_name(works) = UCASE(string3);
1473    3               call SEARCH(M_name(works),works);
1474    3               put skip(2) edit('Enter the work center type :  ')(X(5),A);
1475    3               get edit(string3)(A(80));
1476    3               string3 = UCASE(string3);
1477    3               call OPTIONS(string3);
1478    3               Type(works) = (string3);
1479    3               put skip(2) edit('Enter a capacity :  ')(X(5),A);
1480    3               get edit( string3)(A(80));
1481    3               call lex2(string3,value,err);
1482    3               if( value < 1.0 ) then err = true;
1483    3               else if( value > 127.0 ) then err = true;
1484    3               do while (err );
1485    4
1486    4                       put skip edit('The capacity is out of range.')(X(10),A);
1487    4                       put skip edit('It must be > 0 or  less than 127')(X(10),A);
1488    4                       get edit( string3)(A(80));
1489    4                       call lex2(string3,value,err);
1490    4                       end;
1491    3
1492    3               Capacity(works) = CEIL(value);
1493    3               end;
1494    2
1495    2       end ADDWC;
1496    1
1497 |  1       /*****************************************************************************/
```

```
1499    1       NUKE : proc;
1500    2
1501    2               found = false;
1502    2               do while( ^found);
1503    3
1504    3                       put skip(2) edit('Enter the name of the work center ')(X(5),A);
1505    3                       put skip edit('to be deleted  :  ')(X(5),A);
1506    3                       get edit(string3)(A(80));
1507    3                       string3 = UCASE(string3);
1508    3                       J = 0;
1509    3
1510    3                       do while(^found & J < works);
1511    4
1512    4                               J = J + 1 ;
1513    4                               if( M_name(J) = string3)then found = true;
1514    4                               end;
1515    3
1516    3                       if( ^found ) then put skip edit('Invalid work center name.')(X(5),A);
1517    3                       end;
1518    2
1519    2               do  K = J to (works - 1);
1520    3
1521    3                       Machine(K) = Machine(K+1);
1522    3                       end;
1523    2
1524    2               works = works - 1;
1525    2
1526    2       end NUKE;
1527    1
1528 |  1       /*****************************************************************************/
1529    1
1530    1
1531    1       END_WCEDIT;
1532            EDIT_CNTL : proc;
1533    1
1534    1       dcl     reply   char(80) var;
1535    1       dcl     edgpt   entry,
1536    1               MEP     entry,
1537    1               edpart  entry;
1538    1
1539 |  1       /************************************/
1540    1
1541    1
1542    1       DO while (reply ^= '4');
1543    2
1544    2               call menu;
1545    2               put skip(2) edit('Which module would you like to edit ?  ')(X(16),A);
1546    2               get edit(reply)(A(80));
1547    2               if( reply = '1')then call edgpt;
1548    2               else
1549    2                   if(reply = '2')then call MEP;
1550    2                   else
1551    2                       if(reply = '3')then call edpart;
1552    2                       else if( reply ^= '4')then put skip edit('Invalid option.')(X(16),A);
1553    2
1554    2               end;
1555    1       ...........................................
```

```
1556    1
1557    1       menu : proc;
1558    2       put skip(30) edit('Option          Module')(X(15),A);
1559    2       put skip( 2) edit('  1       General Project Tittles')(X(15),A);
1560    2       put skip( 2) edit('  2       Model Execution Parameters')(X(15),A);
1561    2       put skip( 2) edit('  3       Part Information           ')(X(15),A);
1562    2       put skip( 2) edit('  4       Exit ')(X(15),A);
1563    2       put skip ;
1564    2
1565    2       end menu;
1566    1       /************************/
1567    1       end EDIT_CNTL;
1568            edgpt : proc;
1569    1
1570    1       dcl  reply      char(80) var,   newstring       char(80) var;
1571    1       dcl  Mname      char(80) var ext,
1572    1            Pname      char(80) var ext,
1573    1            date1      char(80) var ext;
1574    1
1575    1       do while ( reply ^= '4' );
1576    2
1577    2
1578    2
1579    2       put skip(25) edit('Item           Parameter')(X(20),A);
1580    2       put skip( 2) edit(' 1           Modelers name')(X(20),A);
1581    2       put skip( 2) edit(' 2           Project  name')(X(20),A);
1582    2       put skip( 2) edit(' 3           Current date ')(X(20),A);
1583    2       put skip( 2) edit(' 4           Exit         ')(X(20),A);
1584    2       put skip(2) edit('Which parameter do you wish to change ?  ')(X(15),A);
1585    2       get edit(reply)(A(80));
1586    2
1587    2       if( reply = '1') then do;
1588    3
1589    3               put skip(3) edit('Enter the new Modelers name : ')(X(10),A);
1590    3               get edit(newstring)(A(80));
1591    3               Mname = newstring;
1592    3               end;
1593    2
1594    2       else if( reply = '2') then do;
1595    3
1596    3               put skip(3) edit('Enter the new Project name :  ')(X(10),A);
1597    3               get edit(newstring)(A(80));
1598    3               Pname = newstring;
1599    3               end;
1600    2
1601    2       else if( reply = '3')then do;
1602    3
1603    3               put skip(3) edit('Enter the new date :  ')(X(10),A);
1604    3               get edit(newstring)(A(80));
1605    3               date1 = newstring;
1606    3               end;
1607    2
1608    2       end;
1609    1
1610    1       end edgpt;
1611
```

```
1613    1
1614    1       dcl     name            char(30) var,   position                fixed bin(7);
1615    1               found           bit(1),         J                       fixed bin(7);
1616    1
1617    1       %include  'MACHINE.PLI';
1641    1
1642    1       %INCLUDE  'UCASE.PLI';
1652    1
1653    1       found = '0'B;
1654    1       do while ( ^found );
1655    2
1656    2               J = 0;
1657    2
1658    2               do while( J < works  & ^found );
1659    3
1660    3                       J = J + 1;
1661    3                       if( M_name(J) = name )then do;
1662    4                               position = J ;
1663    4                               found = '1'B;
1664    4                               end;
1665    3                       end;
1666    2
1667    2
1668    2               if( ^found )then do;
1669    3
1670    3                               put skip(3) edit(name,'is not a valid work center name.')(X(5),A,A);
1671    3                               put skip edit('Re-enter :  ')(X(5),A);
1672    3                               get edit(name)(A(30));
1673    3                               name = UCASE(name);
1674    3                               end;
1675    2
1676    2               end;
1677    1
1678    1       end Look;
1679
1680
1681 |      /*************************************************/
1682
1683            EDIT_MODEL : proc;
1684    1
1685    1       dcl  reply  char(80) var;
1686    1       dcl  WCEDIT  entry,
1687    1            SHTEDIT ENTRY,
1688    1            MHEDIT  ENTRY;
1689    1
1690    1       call esenu;
1691    1
1692    1       reply = '';
1693    1       do while( reply ^= '4');
1694    2
1695    2                               PUT SKIP EDIT('Which module do you wish to edit? ')(X(14),A);
1696    2               get edit( reply ) (A(80));
1697    2               if( reply = '1') then call WCEDIT;
1698    2               else if( reply = '2')then call SHTEDIT;
1699    2               else if( reply = '3')then call MHEDIT;
1700    2               else if( reply ^= '4')then put skip edit('Invalid reply')(X(17),A);
```

```
2               end;
1
1       emenu : proc;
2
2       put skip (25) ;
2       put skip edit('|||||||||||||||||||||||||||||||||||||||||||||||||||||||')(X(14),A);
2       PUT SKIP EDIT('|                                                     |')(X(14),A);
2       PUT SKIP EDIT('|               MODEL STATEMENT EDITOR                |')(X(14),A);
2       PUT SKIP EDIT('|                                                     |')(X(14),A);
2       PUT SKIP EDIT('|    OPTION             FUNCTION                      |')(X(14),A);
2       PUT SKIP EDIT('|                                                     |')(X(14),A);
2       PUT SKIP EDIT('|       1.        EDIT WORK CENTERS                   |')(X(14),A);
2       PUT SKIP EDIT('|       2.        EDIT SHUTTLES                       |')(X(14),A);
2       PUT SKIP EDIT('|       3.        EDIT MATERIAL HANDLING PATHS        |')(X(14),A);
2       PUT SKIP EDIT('|       4.        EXIT                                |')(X(14),A);
2       PUT SKIP EDIT('|                                                     |')(X(14),A);
2       PUT SKIP EDIT('|||||||||||||||||||||||||||||||||||||||||||||||||||||||')(X(14),A);
2
2       end emenu;
1
1       end EDIT_MODEL;

        /****************************/
        /*****************************/
        /******************************/

        SHTEDIT :  PROC;
1
1       %include 'SHUTS.PLI';
1       %INCLUDE 'UCASE.PLI';
1
1       dcl    reply    char(80) var,  newshut    entry,
1              modshut entry        ,  nukeshut   entry;
1
1       call shutmenu;
1
1       reply = '';
1       do while ( reply ^= '4');
2
2               put skip(2) edit('which function would you like to perform? ')(X(18),A);
2               get edit(reply)(A(80));
2               if( reply = '1')then call newshut;
2               else if(reply = '2')then call modshut;
2               else if(reply = '3')then call nukeshut;
2               else if(reply ^= '4')then
2
2                       put skip edit('Invalid reply')(X(18),A);
2               end;
1
1       shutmenu : proc;
2
2       put skip (25);
2       put skip edit('//////////////////////////////////')(x(20),A);
2       PUT SKIP EDIT('/                                /')(X(20),A);
2       PUT SKIP EDIT('/         SHUTTLE EDITOR         /')(X(20),A);
2       PUT SKIP EDIT('/                                /')(X(20),A);
```

```
 1791    2       PUT SKIP EDIT('/                                    /')(X(20),A);
 1792    2       PUT SKIP EDIT('/    1.      ADD A SHUTTLE           /')(X(20),A);
 1793    2       PUT SKIP EDIT('/    2.      MODIFY A SHUTTLE        /')(X(20),A);
 1794    2       PUT SKIP EDIT('/    3.      DELETE A SHUTTLE        /')(X(20),A);
 1795    2       PUT SKIP EDIT('/    4.      EXIT                    /')(X(20),A);
 1796    2       PUT SKIP EDIT('/                                    /')(X(20),A);
 1797    2       PUT SKIP EDIT('//////////////////////////////////////')(X(20),A);
 1798    2
 1799    2       end shutmenu;
 1800    1
 1801    1       end SHTEDIT;
 1802            /************************/
 1803            /************************/
 1804            /************************/
 1805
 1806            newshut : proc;
 1807    1
 1808    1       dcl  lex2  entry ( char(80) var, float bin(31),bit(1)),
 1809    1            Que   entry ( char(80) var, fixed bin(7)),
 1810    1            Parm_in entry ( char(80) var,fixed bin(7));
 1811    1
 1812    1       %include 'UCASE.PLI';
 1822    1       %INCLUDE 'SHUTS.PLI';
 1846    1
 1847    1       dcl    more  bit(1), true  bit(1) init('1'B), false bit(1) init('0'B),
 1848    1              string1  char(80) var,string2  char(80) var, string3 char(80) var,
 1849    1              string4  char(80) var, string5  char(80) var, string6 char(80) var,
 1850    1              string7  char(80) var, caps  float bin(31), numerr bit(1),
 1851    1            ioerr   bit(1), pointer  fixed bin(7);
 1852    1
 1853    1
 1854    1       /*****************************************************************************/
 1855    1
 1856    1
 1857    1         put skip(5) list('Do you need to see the instruction set? ( YES/NO ): ');
 1858    1         get list(string1);
 1859    1         string1 = UCASE(string1);
 1860    1
 1861    1         if(string1 = 'YES' | string1 = 'Y' ) then call instructions;
 1862    1
 1863    1
 1864    1           put skip(2) list('Shuttle name: ');
 1865    1           get edit(string1) ( A(30));
 1866    1           string1 = UCASE(string1);
 1867    1
 1868    1           do while (string1 ^=  'QUIT');
 1869    2
 1870    2                    SHUTS = SHUTS + 1 ;
 1871    2
 1872    2
 1873    2              C_name(SHUTS) = string1;
 1874    2
 1875    2              put skip(2) list('Capacity , Q Rank , Q number , Work Center , I/O , Distribution ');
 1876    2
 1877    2              put skip(2) edit('>') ( A,X(5));
 1878    2              get list(string2,string3,string7,string4,string5,string6);
```

```
1880    2               numerr = false;
1881    2               Call lex2(string2,caps,numerr);
1882    2
1883    2               do while ( numerr );
1884    3
1885    3                  put skip(2) list('The capacity must be numeric, re-enter: ');
1886    3                  get list(string2);
1887    3
1888    3                  numerr = false;
1889    3                  Call lex2(string2,caps,numerr);
1890    3                  end;
1891    2
1892    2             Nums(SHUTS) = CEIL(caps);
1893    2
1894    2           numerr = false;
1895    2
1896    2             call lex2(string7,caps,numerr);
1897    2
1898    2                    if( caps > 100 | caps < 1 ) then numerr = '1'B;
1899    2
1900    2           do while(numerr);
1901    3
1902    3              put skip(2) list('The Q number is invalid, re-enter:  ');
1903    3              get edit(string7)(A(80));
1904    3              numerr = false;
1905    3              call lex2( string7,caps,numerr);
1906    3
1907    3                 if(  caps > 100 | caps < 1 ) then numerr = '1'B;
1908    3
1909    3              end;
1910    2
1911    2
1912    2
1913    2
1914    2                 pointer = CEIL( caps );
1915    2
1916    2             string3 = UCASE(string3 );
1917    2             Call QUE(string3,pointer );
1918    2
1919    2             Que_num(SHUTS) = pointer;
1920    2             Link(SHUTS) = UCASE(string4);
1921    2
1922    2             string5 = UCASE(string5);
1923    2             IOerr = true;
1924    2             if( string5 = 'I'  | string5  = 'O' ) then IOerr = false;
1925    2             do while ( IOerr );
1926    3
1927    3                put skip(2) list('The shuttle was not specified as Input or Output !');
1928    3                put skip list('Re-enter: ');
1929    3                get list(string5);
1930    3                   string5 = UCASE(string5);
1931    3                if( string5  = 'I' | string5  = 'O' )then IOerr = false;
1932    3                end;
1933    2
1934    2             IO(SHUTS)  =  string5  ;
1935    2
1936    2             string4 = UCASE( string4 );
```

```
1937    2
1938    2           Call  Parm_in(string6,pointer);
1939    2           Moves(SHUTS) = pointer;
1940    2
1941    2           M_dist( SHUTS ) = string6 ;
1942    2
1943    2           put skip(2) list('Shuttle name:  ');
1944    2           get list( string1 );
1945    2           string1 = UCASE(string1);
1946    2
1947    2       End;  /* end of QUIT Loop  */
1948    1
1949 |  1   /*************************************************************/
1950    1
1951    1
1952    1   instructions : proc;
1953    2
1954    2   put skip(3) edit('####  Shuttle Input Instructions  ####')(X(20),A);
1955    2   put skip(3) edit('The shuttle input description is divided into two')(X(20),A);
1956    2   put skip edit('sections.  The first section asks you for a shuttle')(X(15),A);
1957    2   put skip edit('name.  You may either input a name less than 30')(X(15),A);
1958    2   put skip edit('characters or type QUIT to terminate the shuttle ')(X(15),A);
1959    2   put skip edit('input sequence.')(X(15),A);
1960    2   put skip edit('The second section will ask you to input six ')(X(20),A);
1961    2   put skip edit('different items.  The six input items are as ')(X(15),A);
1962    2   put skip edit('follows:  Shuttle Capacity, Queue Rank, Queue Disipline, ')(X(15),A);
1963    2   put skip edit('Work Center link, Shuttle funciton: input/output, and a ')(X(15),A);
1964    2   put skip edit('distribution name for movement times.  The prompt for ')(X(15),A);
1965    2   put skip edit('the second section of input is a " > " .')(X(20),A);
1966    2
1967    2
1968    2   end instructions;
1969    1
1970 |  1   /*********************************************************************/
1971    1   end newshut;
1972
1973 |      /*********************************/
1974
1975      nukeshut : proc;
1976    1
1977    1           %INCLUDE 'UCASE.PLI'; %INCLUDE 'SHUTS.PLI';
2010    1
2011    1
2012    1   dcl  no bit(1), true bit(1) init('1'B),false bit(1) init('0'B),
2013    1        J  fixed bin(7), found bit(1), string char(80) var,
2014    1        reply char(80) var, K  fixed bin(7);
2015    1
2016    1   dcl  getname   entry( fixed bin(7));
2017    1
2018    1
2019    1   no = true;
2020    1
2021    1   do while( no);
2022    2
2023    2           call getname(J);
2024    2
```

```
2026    2       found = false;
2027    2       do while( ^found);
2028    3
2029    3               put skip(2) edit('You have chosen to delete the shuttle ',C_name(J))(X(5),A,A);
2030    3               put skip edit('Is it alright to delete ?','Reply( YES/NO )')(X(5),A,skip,X(5),A);
2031    3               get edit(reply)(A(80));
2032    3               reply = UCASE(reply);
2033    3               if( reply = 'YES' | reply = 'NO')then found = true;
2034    3               else put skip edit('Invalid reply')(X(5),A);
2035    3
2036    3               if( reply = 'YES')then no= false;
2037    3               end;
2038    2
2039    2               end;
2040    1
2041    1       do K = J to SHUTS-1;
2042    2
2043    2               CARTS(K) = CARTS(K+1);
2044    2               END;
2045    1
2046    1       end nukeshut;
2047
2048 |          /*****************************************/
2049
2050          modshut : proc;
2051    1
2052    1       dcl reply   char(80) var,  J  fixed bin(7), found bit(1),
2053    1           true  bit(1) init('1'B), false  bit(1) init('0'B),
2054    1           name   char(80) var,  err bit(1);
2055    1
2056    1       %INCLUDE 'UCASE.PLI';  %INCLUDE 'SHUTS.PLI';
2089    1
2090    1       dcl  Parm_in  entry( char(80) var,fixed bin(7)),
2091    1            Que      entry(char(80) var,fixed bin(7)),
2092    1            lex2     entry(char(80) var,float bin(31),bit(1));
2093    1
2094    1
2095    1
2096    1
2097    1       call scmenu;
2098    1       do while( reply ^= '6');
2099    2
2100    2       put skip edit('Which function would you like to perform ? ')(X(15),A);
2101    2       get edit( reply)(A(80));
2102    2       if( reply = '1')then call newname;
2103    2       else if(reply = '2')then call newtype;
2104    2       else if(reply = '3')then call newlink;
2105    2       else if(reply = '4')then call newdist;
2106    2       else if(reply = '5')then call newq;
2107    2       else if(reply ^= '6')then
2108    2
2109    2               put skip edit('Invalid reply.')(X(15),A);
2110    2
2111    2               end;
2112    1
2113    1
```

```
2213     2
2214     2      end newtype;
2215     1
2216   ! 1      /*******************************/
2217     1
2218     1      newlink : proc;
2219     2
2220     2      dcl  J   fixed bin(7),  string char(80) var;
2221     2      dcl  getname  entry( fixed bin(7));
2222     2
2223     2      %include 'UCASE.PLI';
2233     2      %INCLUDE 'SHUTS.PLI';
2257     2
2258     2      call getname(J);
2259     2      put skip edit('Enter the new work center link : ')(X(10),a);
2260     2      get edit( string )(A(80));
2261     2      string = UCASE(string);
2262     2      Link(J) = string ;
2263     2
2264     2      end newlink;
2265     1
2266   ! 1      /*********************************/
2267     1
2268     1      newdist : proc;
2269     2
2270     2      %include 'UCASE.PLI';
2280     2      %INCLUDE 'SHUTS.PLI';
2304     2
2305     2      dcl  Parm_in  entry( char(80) var, fixed bin(7));
2306     2      dcl  getname  entry(fixed bin(7));
2307     2
2308     2      dcl (J,pointer)fixed bin(7);
2309     2      dcl string     char(80) var;
2310     2
2311     2      call getname(J);
2312     2      put skip edit('Enter the new distribution name :  ')(X(10),a);
2313     2      get edit(string)(A(80));
2314     2      string = UCASE(string);
2315     2      call Parm_in(string,pointer);
2316     2
2317     2      M_dist(J) = string;
2318     2      Moves(J) = pointer;
2319     2
2320     2      end newdist;
2321     1
2322   ! 1      /*******************************************/
2323     1      newq : proc;
2324     2
2325     2      dcl   (J,pointer) fixed bin(7);
2326     2      dcl  value  float bin(31);
2327     2      dcl  err bit(1);
2328     2      dcl getname  entry(fixed bin(7)),
2329     2          lex2     entry(char(80) var,float bin(31),bit(1));
2330     2
2331     2      dcl  string  char(80 ) var;
2332     2
2333     2      call getname(J);
```

```
put skip(25);
PUT SKIP EDIT('\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\')(X(15),A);
PUT SKIP EDIT('\                                      \')(X(15),A);
PUT SKIP EDIT('\          SHUTTLE MODIFICATIONS       \')(X(15),A);
PUT SKIP EDIT('\                                      \')(X(15),A);
PUT SKIP EDIT('\   OPTIONS          FUNCTION          \')(X(15),A);
PUT SKIP EDIT('\     1.      CHANGE A SHUTTLE NAME    \')(X(15),A);
PUT SKIP EDIT('\     2.      CHANGE A SHUTTLE TYPE    \')(X(15),A);
PUT SKIP EDIT('\     3.      CHANGE A WORK CENTER LINK \')(X(15),A);
PUT SKIP EDIT('\     4.      CHANGE A MOVEMENT DIST   \')(X(15),A);
PUT SKIP EDIT('\     5.      CHANGE THE QUEUE LINK    \')(X(15),A);
PUT SKIP EDIT('\     6.      EXIT                     \')(X(15),A);
PUT SKIP EDIT('\                                      \')(X(15),A);
PUT SKIP EDIT('\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\')(X(125),A);

end scmenu;


/************************/

newname : proc;

%include 'UCASE.PLI';

dcl  name   char(80) var,  found bit(1), J fixed bin(7);

dcl  getname  entry(fixed bin(7));

call getname(J);
put skip edit('Enter the new name : ')(X(5),A);
get edit(name)(A(80));
name = UCASE(name);
C_name(J) = name;
end newname;

/**************************/
newtype : proc;

%include 'SHUTS.PLI';
%INCLUDE 'UCASE.PLI';

DCL  J FIXED BIN(7), err bit(1),  string char(80) var;
dcl  getname entry(fixed bin(7));

call getname(J);
err = '1'B;
do while( err );

        put skip edit('Enter the new type , Input or Output : ')(X(10),A);
        put skip edit('Reply( I/O ) : ')(X(10),A);
        get edit( string)(A(80));
        string = UCASE(string);
        if( string = 'I'  | string = 'O' ) then err = '0'B;
        end;
```

```
2334     2       err = '1'B;
2335     2       do while(err );
2336     3
2337     3               put skip edit('Enter the new queue number :  ')(X(10),A);
2338     3               get edit(string)(A(80));
2339     3               call lex2(string,value,err);
2340     3               if( value < 1.0 | value > 100.0 ) then do;
2341     4
2342     4                       err = '1'B;
2343     4                       put skip edit('Invalid queue number ')(X(10),a);
2344     4                       end;
2345     3
2346     3               else err = '0'B;
2347     3               end;
2348     2
2349     2       pointer = CEIL(value);
2350     2       call Que('',pointer);
2351     2
2352     2       Que_num(J) =  pointer;
2353     2
2354     2       end newq;
2355     1
2356     1
2357     1       end modshut;
2358
2359
2360 |           /*****************************************/
2361
2362             getname : proc(J);
2363     1
2364     1       dcl  J  fixed bin(7);
2365     1
2366     1       dcl  found bit(1);
2367     1       dcl  name char(80) var;
2368     1
2369     1       %include 'UCASE.PLI';
2379     1       %include 'SHUTS.PLI';
2403     1
2404     1       found = '0'B;
2405     1
2406     1       do while( ^found);
2407     2
2408     2               put skip edit('Enter the name of the shuttle you wish to change :  ')(X(5),A);
2409     2               get edit(name)(A(80));
2410     2               name = UCASE(name);
2411     2
2412     2       J = 0;
2413     2       do while( J < SHUTS & ^found);
2414     3
2415     3               J = J + 1;
2416     3               if(C_name(J) = name ) then found = '1'B;
2417     3               end;
2418     2
2419     2               IF( ^found ) then put skip edit('Invalid shuttle name.')(X(10),a);
2420     2
2421     2       end;
```

```
                /*  This procedure prompts the user for the input of
                    the work center descriptions.  It is called by
                    the procedure MODEL.


                */

                MACH :proc;
     1
     1
     1          %INCLUDE "MACHINE.PLI";
     1
     1
     1
     1
     1
     1            %INCLUDE  "UCASE.PLI";
     1
     1           dcl  new bit(1),
     1                old bit(1);
     1
     1
     1          dcl
     1             string1   char(80) var,
     1             string2   char(80) var,
     1             string3   char(80) var;
     1
     1          dcl ERROR   entry (fixed bin(31),char(*) var );
     1
     1          dcl OPTIONS  entry  (char (80) var);
     1
     1          dcl  MACH_IN    entry;
     1
     1          dcl  LOGIC      entry;
     1
     1          dcl  SAVE_MACH    entry;
     1
     1          dcl lex2  entry (char(80) var,float bin(31),bit(1));
     1
     1          dcl ERRCHK  entry(float bin(31),float bin (31),bit(1),fixed bin(31));
     1
     1          dcl  SEARCH  entry (char(30) var,fixed bin(15));
     1
     1          dcl arr bit(1);
     1          dcl value float bin(31);
     1
     1          dcl II    fixed bin(15) ;
     1
     1          /*****************************************************/
     1
     1          put skip(25);
     1          put skip edit( "****  WORK CENTER DESCRIPTIONS  ****") (X(20),A);
```

```
 88     1
 89     1      old = '0'B;   new = '0'B;
 90     1
 91     1
 92     1
 93     1       err = '1'B;
 94     1
 95     1        do while ( err );
 96     2
 97     2        put skip(3) list('Do you wish to use NEW or EXISTING Work Center Descriptions ?');
 98     2        put skip list('Reply ( NEW/OLD ): ');
 99     2        get list( string1);
100     2
101     2           string1 = UCASE( string1 );
102     2
103     2           if ( string1  = 'NEW' | string1  = 'OLD')then err = '0'B;
104     2
105     2           else
106     2                   err = '1'B;
107     2          end;
108     1
109     1         if( string1 = 'NEW') then new = '1'B;
110     1
111     1          else
112     1                   old = '1'B;
113     1
114     1
115     1          do while ( new );  /*  Create new work center descriptions  */
116     2
117     2       put skip(5) list('Do you need to see the instruction set ?(YES/NO) ');
118     2       get list(string1);
119     2       string1 = UCASE(string1);
120     2
121     2       if(string1 = 'YES' | string1 = 'Y' ) then call instructions;
122     2
123     2
124     2             new = '0'B;
125     2
126     2
127     2        err = '1'B;
128     2
129     2        do while (err);
130     3
131     3       Put skip(2)  edit('How many work centers are there in your system ?') (X(5),A);
132     3       get list(string1);
133     3
134     3       err = '0'B;
135     3       Call lex2(string1,value,err);
136     3
137     3       if ( value <= 0.0 ) then do;
138     4                                   call ERROR ( 110,string1 );
139     4                                   err = '1'B;
140     4                                   end;
141     3
142     3         else
143     3
144     3
```

```
  145     3        if( value > 25.0 ) then do;
  146     4                                  put skip list('You are limited to 25 work centers.');
  147     4                                  put skip list('Please re-enter the number of work centers.');
  148     4                                  err = '1'B;
  149     4                                 end;
  150     3
  151     3          else
  152     3
  153     3       if (^err) then
  154     3         works = CEIL(value);
  155     3
  156     3       end;
  157     2
  158     2
  159     2         do  II = 1 to works by 1;
  160     3
  161     3            err = '1'B;
  162     3
  163     3            do while ( err );
  164     4
  165     4                put skip(2) edit('Work Center ',II,'      ')(A,F(2),A);
  166     4                get list ( string1,string2,string3);
  167     4
  168     4               string1 = UCASE(string1);
  169     4               string3 = UCASE(string3);
  170     4
  171     4
  172     4            M_name ( II ) = string1;
  173     4
  174     4
  175     4       if( II > 1 )then
  176     4                Call  SEARCH(M_name(II),II);
  177     4
  178     4                err = '0'B;
  179     4                Call lex2(string2,value,err);
  180     4
  181     4                        if( value < 1.0 ) then
  182     4
  183     4                                err = '1'B;
  184     4
  185     4                      do while ( err ) ;
  186     5
  187     5                       put skip list('The work center capacity must be > 0 ');
  188     5                       put skip list('Re-enter :  ');
  189     5                       get edit (string2)(A(80));
  190     5                       err = '0'B;
  191     5                       call lex2(string2,value,err);
  192     5                       if( value < 1.0) then err = '1'B;
  193     5                       end;
  194     4
  195     4
  196     4                       Capacity ( II ) = FLOOR ( value );
  197     4
  198     4                if (^err ) then call OPTIONS(string3);
  199     4
  200     4
  201     4
```

```
                    Type ( II ) = string3;
               end; /*  end of individual machine input and error checks  */


     end;  /* end of mach loop */
           CALL   LOGIC  ;

     err ='1'B;
     do while ( err );
           put skip(5) list('Would you like the Work Center descriptions saved in a file ? ( Y/N ): ');
           get list(string1);
           string1 = UCASE( string1 );
           if(string1 = 'Y'  | string1 = 'N') then err = '0'B;
            end;
      if(string1 = 'Y' ) then call SAVE_MACH;
        END; /* End of  "new" loop  */
        do while ( old );
             old = '0'B;
                Call  MACH_IN;
             END;


     /*************  Internal procedures  ***************/

     instructions : proc;

     put skip(3) edit('****  Work Center Input Instructions  ****') (X(15),A);

     put skip(3) list('     The first item you will be asked for is the number');
     put skip list('of machine centers in the system.  The number must');
     put skip list('be less then 25.   Next you will be asked to input');
     put skip list('the description of each work center in the system one');
     put skip list('at a time.  The input format consists of three fields.');
     put skip list('The first field is the name of the work center. The ');
     put skip list('second field is the capacity of the work center. ');
     put skip list('The capacity is the number of parts that can be ');
     put skip list('processed in parallel by the work center.  Finally,');
     put skip list('the third field is the type of work center.  The');
     put skip list('different types are listed below.');
     put skip(2) list('     Work Center Types');
     put skip(2) list('     WC     => Regular work center either a machine or manual station');
     put skip list   ('      L     => Load station');
     put skip list   ('     UL     => Unload station');
```

```
259     2
260     2       end instructions;
261     1
262     1
263     1       end MACH;
264     |       /*
265     |          This procadure searchas the structure Machine
266     |       for the given work center name.  If a duplicate is
267     |       found tha user is promptad to change either name.
268     |
269     |       */
270
271             SEARCH : proc ( name,j);
272     1       % INCLUDE 'UCASE.PLI';
282     1
283     1
284     1       dcl   ERROR  entry  (fixed bin(31),char(80) var);
285     1       dcl   lex2   entry  (char(80) var,float bin(31),bit(1));
286     1
287     1
288     1       %INCLUDE  'MACHINE.PLI';
312     1
313     1
314     1
315     1       dcl name char(30) var;
316     1       dcl wnum fixed bin(31);
317     1       dcl (j,k)fixed bin(15);
318     1
319     1       dcl ( duplicate,error1)  bit(1);
320     1       dcl (e,err) bit(1) init('0'B);
321     1       dcl more bit(1) init('1'B);
322     1       dcl wcnum  char(80) var;
323     1       dcl val float bin(31);
324     1
325     1
326     1
327     1       duplicate = '1'B;
328     1
329     1         do while( duplicate );
330     2
331     2            error1 = '0'B;
332     2            k = 0;
333     2
334     2            do while ( k < (j-1)  & ( ^error1 ) );
335     3
336     3               k =k + 1;
337     3
338     3               if( M_name( k ) = name ) then error1 = '1'B;
339     3
340     3               end;
341     2
342     2           if( error1 ) then call new_name; /* internal procedure to prompt
343     |  2                                             for a new work center name
344     |  2                                          */
345     2
346     2            else  duplicate = '0'B;
347     2
```

```
348     2           end;
349     1
350   | 1       /*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
351     1
352     1       new_name : proc;
353     2
354     2       put skip(2) edit("######  ","Duplicate Work Center Name" ) (A,X(5),A);
355     2       put skip(2)    edit("Work Center ",k," has the same name") (A,F(2),A);
356     2       put skip edit("as Work Center ",j)(A,F(2));
357     2
358     2        err = "1"B;
359     2
360     2
361     2          put skip(2) edit("Enter a new name for Work Center ",j," ")
362     2                      (A,F(2),A);
363     2
364     2
365     2         get list(name);
366     2         name = UCASE(name);
367     2
368     2       End new_name;
369     1
370     1       end SEARCH;
371             MACH_IN  : proc;
372     1
373     1       %include  "PAUSE.PLI";
382     1
383     1
384     1
385     1       %INCLUDE  "UCASE.PLI";
395     1
396     1       %INCLUDE  "FILETEST.PLI";
417     1
418     1
419     1       %INCLUDE  "MACHINE.PLI";
443     1
444     1
445     1       dcl  1   RECORD    ,
446     1
447     1                  2  name  char(30),
448     1                  2  name2 char(10),
449     1                  2  number   fixed bin(7);
450     1
451     1       dcl  symbol char(3) var;
452     1
453     1       dcl position fixed bin(7);
454     1
455     1
456     1
457     1         dcl  alpha  char(30) var  init("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
458     1
459     1
460     1
461     1
462     1       dcl  error   bit(1),
463     1            true    bit(1) init("1"B),
464     1            false   bit(1) init("0"B);
```

```
  465     1
  466     1
  467     1       dcl  more_records   bit(1);
  468     1
  469     1       dcl  reply     char(80) var,
  470     1            filename  char(12)  var;
  471     1
  472     1
  473     1       dcl  racnum     fixed bin(7),
  474     1            records    fixed bin(7),
  475     1            JJ         fixed bin(7);
  476     1
  477     1       dcl   logical  file   input   keyed sequential
  478     1
  479     1       environment( maximum_record_number(50),maximum_record_size(250));
  480     1
  481     1
  482     1       on  endfile(logical) more_records = false;
  483     1       more_records = true;
  484     1
  485     1       on undefinedfile  ( logical ) begin;
  486     2
  487     2          put skip(2) edit('The file',filename,'does not exist.')(A,X(1),A(12),X(1),A);
  488     2
  489     2          error = true;
  490     2
  491     2          end;
  492     1
  493     1
  494     1       ON ANYCONDITION   BEGIN;
  495     2
  496     2                  Put skip list('Input/Output Error');
  497     2                   error = true;
  498     2                  end;
  499     1
  500     1
  501     1
  502     1
  503   | 1       /*****************************************************/
  504     1
  505     1
  506     1       put skip(25) edit('****  FMSS Filing System  ****')(X(22),A);
  507     1
  508     1       error = true;
  509     1
  510     1       do while( error );
  511     2
  512     2         put skip(3) edit('Do you need instructions ?( Y/N ): ') (X(20),A);
  513     2
  514     2         get edit(reply) (A(80));
  515     2         reply = UCASE(reply);
  516     2
  517     2         if( reply = 'Y' | reply = 'N')then error = false;
  518     2
  519     2          end;
  520     1
  521     1          if(reply = 'Y') then call instructions;
```

```
522  1
523  1          error = true;
524  1
525  1          do while( error );
526  2
527  2            put skip(5) edit('Enter a valid file name: ') (X(25),A);
528  2            get edit(filename) ( A(8));
529  2
530  2             filename = UCASE( filename );
531  2
532  2
533  2
534  2                    symbol = SUBSTR(filename,1,1);
535  2                    position = INDEX(alpha,symbol);
536  2                    if ( position = 0 ) then do;
537  3
538  3                            put skip(2) edit('The file ',filename,' is invalid, re-enter!')( A,A,A);
539  3
540  3                     error = true;
541  3                     end;
542  2
543  2
544  2                 else   error = false;
545  2
546  2                    open file ( logical )  title( filename );
547  2
548  2
549  2
550  2                        recnum = 0 ;
551  2                        read file(logical)  into (RECORD)  keyto(recnum);
552  2
553  2                     if(name ^= 'Work Center Descriptions')then do;
554  3
555  3                            put skip edit('The file',filename,'does not contain Work Center Descriptions.')(A,)
556  3                            put skip list('You will need to enter a different filename.');
557  3                            error = true;
558  3                            Close file( logical ) ;
559  3                            end;
560  2
561  2          end;/* end of error  do while  */
562  1
563  1
564  1
565  1                  error = false;
566  1                  put skip(5) edit('The Work Center descriptions are being read from ',filename)(X(8),A,A(8));
567  1                  filename = filename || '.DAT' ;
568  1
569  1                    records = 0;
570  1
571  1                  do  while ( more_records );
572  2
573  2
574  2                   records = records + 1 ;
575  2
576  2                   read  file(logical) into ( RECORD )  keyto(recnum);
577  2
578  2                   M_name(records) = name;
```

```
579     2                   Type(records)   = name2;
580     2                   Capacity(records) = number;
581     2
582     2
583     2               end;/* end of write loop  */
584     1
585     1                        works = records - 1;
586     1
587     1
588     1                put skip(2) edit('The file has been created.') ( X(24),A);
589     1
590     1                     put skip edit('There were',works,'records read from the file.')(X(16),A,X(1),F(2),X(1),A);
591     1
592     1                    close file ( logical ) ;
593     1
594     1                     Call Pause;
595     1
596     1
597     1
598 |   1       /*********************************************************/
599     1
600     1       instructions : proc;
601     2
602     2       put skip(5) edit('++++  Filing System Instructions  ++++') ( X(21),A);
603     2       put skip(3) edit('In the filing system you will be asked to enter') (X(15),A);
604     2       put skip edit('a file name where your model statements can be')(X(10),A);
605     2       put skip edit('saved.  The file name can consist of from 1 to 8')(X(10),A);
606     2       put skip edit('letters or numbers.  However, the first character') (X(10),A);
607     2       put skip edit('must be a letter.  Examples of valid file names')(X(10),A);
608     2       put skip edit('are shown below.') ( X(10),A);
609     2       put skip(2) edit('Valid File Names')(X(29),A);
610     2       put skip(2) edit('USERFILE ')(X(33),A);
611     2       put skip    edit('MACHINE2 ')(X(33),A);
612     2       put skip    edit('FILE2326 ')(X(33),A);
613     2       put skip    edit('F1234567' )(X(33),A);
614     2
615     2
616     2       END instructions;
617     1
618     1
619     1       END  MACH_IN;
620             SAVE_MACH  :  proc;
621     1
622     1
623     1       %include 'PAUSE.PLI';
632     1
633     1       %INCLUDE  'UCASE.PLI';
643     1
644     1       %INCLUDE  'FILETEST.PLI';
665     1
666     1
667     1       %INCLUDE  'MACHINE.PLI';
691     1
692     1
693     1       dcl  1   RECORD    ,
694     1
695     1                 2  name  char(30),
```

```
  696    1                 2  name2 char(10),
  697    1                 2  number   fixed bin(7);
  698    1
  699    1       dcl  symbol char(3) var;
  700    1
  701    1       dcl position fixed bin(7);
  702    1
  703    1
  704    1
  705    1         dcl  alpha  char(30) var  init('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  706    1
  707    1
  708    1
  709    1
  710    1       dcl  error   bit(1),
  711    1            true    bit(1) init('1'B),
  712    1            false   bit(1) init('0'B);
  713    1
  714    1       dcl  reply     char(80) var,
  715    1            filename  char(12)  var;
  716    1
  717    1
  718    1       dcl  recnum      fixed bin(7),
  719    1            JJ          fixed bin(7);
  720    1
  721    1       dcl  OUTFILE  file         keyed environment(
  722    1
  723    1                                    maximum_record_number(50),
  724    1                                          maximum_record_size(250));
  725    1
  726    1
  727    1
  728    1
  729    1       /*****************************************************/
  730    1
  731    1
  732    1       put skip(25) edit('####  FMSS Filing System  ####')(X(22),A);
  733    1
  734    1       error = true;
  735    1
  736    1       do while( error );
  737    2
  738    2         put skip(3) edit('Do you need instructions ?( Y/N ): ') (X(20),A);
  739    2
  740    2         get edit(reply) (A(80));
  741    2         reply = UCASE(reply);
  742    2
  743    2         if( reply = 'Y' | reply = 'N')then error = false;
  744    2
  745    2          end;
  746    1
  747    1          if(reply = 'Y') then call instructions;
  748    1
  749    1          error = true;
  750    1
  751    1          do while( error );
  752    2
```

```
753     2           put skip(5) edit('Enter a valid file name: ') (X(25),A);
754     2           get edit(filename) ( A(8));
755     2
756     2            filename = UCASE( filename );
757     2
758     2
759     2
760     2                   symbol = SUBSTR(filename,1,1);
761     2                   position = INDEX(alpha,symbol);
762     2                   if ( position = 0 ) then do;
763     3
764     3                    put skip(2) edit('The file',filename,'is invalid, re-enter! ')(x(18),A,x(1),A,A);
765     3                    error = true;
766     3                    end;
767     2
768     2
769     2              else   error = false;
770     2
771     2         end;/* end of error  do while  */
772     1
773     1
774     1
775     1                 error = false;
776     1                 put skip(5) edit('The Work Center descriptions are being written into ',filename)(X(8),A,A(8));
777     1                 filename = filename || '.DAT' ;
778     1
779     1
780     1
781     1
782     1
783     1                       open file( OUTFILE )  output  title(filename);
784     1
785     1                   recnum = 1;
786     1
787     1
788     1                   name = 'Work Center Descriptions';
789     1                   number = works;
790     1                   write file(OUTFILE)  from(RECORD)   keyfrom(recnum);
791     1
792     1
793     1                 do JJ = 1 to works;
794     2
795     2                    recnum = recnum + 1 ;
796     2             name = M_name( JJ );
797     2             name2 = Type(JJ);
798     2             number = Capacity ( JJ );
799     2
800     2       write file ( OUTFILE ) from (RECORD)  keyfrom(recnum );
801     2
802     2
803     2            end;/* end of write loop  */
804     1
805     1             put skip(2) edit('The file has been created.') ( X(24),A);
806     1
807     1                     put skip edit('There were',works,'records written.')(X(20),A,X(1),F(2),X(1),A);
808     1
809     1
```

```
   810    1                        close file( OUTFILE );
   811    1
   812    1                         call Pause;
   813    1
   814    1
   815    1
   816    1     /*********************************************************/
   817    1
   818    1     instructions : proc;
   819    2
   820    2     put skip(5) edit("++++  Filing System Instructions  ++++") ( X(21),A);
   821    2     put skip(3) edit("In the filing system you will be asked to enter") (X(15),A);
   822    2     put skip edit("a file name where your model statements can be")(X(10),A);
   823    2     put skip edit("saved.  The file name can consist of from 1 to 8")(X(10),A);
   824    2     put skip edit("letters or numbers.  However, the first character") (X(10),A);
   825    2     put skip edit("must be a letter.  Examples of valid file names")(X(10),A);
   826    2     put skip edit("are shown below.") ( X(10),A);
   827    2     put skip(2) edit("Valid File Names")(X(29),A);
   828    2     put skip(2) edit("USERFILE ")(X(33),A);
   829    2     put skip    edit("MACHINE2 ")(X(33),A);
   830    2     put skip    edit("FILE2326 ")(X(33),A);
   831    2     put skip    edit("F1234567 ")(X(33),A);
   832    2
   833    2
   834    2     END instructions;
   835    1
   836    1
   837    1     END  SAVE_MACH;
   838          DOWN   :  proc;
   839    1     dcl    err bit(1),
   840    1            true  bit(1)  init('1'B),
   841    1            false bit(1)  init('0'B);
   842    1
   843    1     dcl    count    fixed bin(7);
   844    1
   845    1     dcl     string1   char(80) var,
   846    1             string2   char(80) var,
   847    1              string3   char(80) var,
   848    1             string4   char(80) var,
   849    1             null      char(80) var init('');
   850    1
   851    1     %INCLUDE  'UCASE.PLI';
   861    1
   862    1     DCL  Parm_in    entry  ( char(80) var,fixed bin(7));
   863    1
   864    1     DCL   1     BREAKS   (100)    external,
   865    1              2  Element      char(30) var,
   866    1              2  F_dist       char(10) var,
   867    1              2  F_num        fixed bin(7),
   868    1              2  D_dist       char(10) var,
   869    1              2  D_num        fixed bin(7),
   870    1              2  description  char(400 ) var,
   871    1              2  Noccurrs     fixed bin(15),
   872    1              2  Sumdur       float bin(31),
   873    1              2  sumsq        float bin(47);
   874    1
   875    1     dcl    NUMBRKS             fixed bin(7)   external;
```

```
876     1
877     1
878     1
879 !   1       /*************************************************/
880     1
881     1
882     1       put skip (25);
883     1
884     1       put skip edit('****  Breakdown Descriptions  ****') (X(20),A);
885     1       put skip (4);
886     1
887     1       err = true;
888     1
889     1       do while (err );
890     2
891     2       put skip list('Do you need to see the input instructions ? ( Y/N ): ');
892     2
893     2       get list (string1);
894     2       string1 = UCASE(string1);
895     2
896     2        if( string1 = 'Y'  | string1 = 'N')then err = false;
897     2
898     2        end;
899     1
900     1
901     1          if(string1 = 'Y') then call instructions;
902     1
903     1       put skip(2) list ('Enter  "QUIT ,,," to end the input mode. ');
904     1       put skip list('Ready for Breakdown Descriptions');
905     1       put skip(2) edit('\',')      ')(A,A);
906     1
907     1       string1 = null;
908     1
909     1          count = 0;
910     1
911     1          get list(string1,string2,string3);
912     1            string1 = UCASE(string1);
913     1
914     1                do while( string1  ^=  'QUIT');
915     2
916     2
917     2              count = count + 1 ;
918     2
919     2                  Element(count) = string1;
920     2
921     2                    string2 = UCASE(string2);
922     2
923     2                    Call Parm_in(string2,F_num(count));
924     2                    F_dist(count) = string2;
925     2
926     2                    string3 = UCASE(string3);
927     2                    Call Parm_in(string3,D_num(count));
928     2
929     2                       D_dist(count) = string3;
930     2
931     2
932     2                    err = true;
```

```
933     2
934     2                  do while ( err );
935     3
936     3                    put skip(2) list('Description of breakdown ? ( Y/N ): ');
937     3                    get list(string4);
938     3               string4 = UCASE(string4);
939     3
940     3
941     3                  if( string4 = 'Y'  |  string4  = 'N' )then err = false;
942     3
943     3                   end;
944     2
945     2             if( string4 = 'Y' ) then   call  SCENARIO ;
946     2
947     2
948     2
949     2          put skip(2) list('New breakdown entry :');
950     2                Put skip Edit('\','      ')  ( A , A );
951     2
952     2
953     2                get list(string1,string2,string3);
954     2                string1 = UCASE(string1);
955     2
956     2
957     2             end;
958     1
959     1
960     1          NUMBRKS =  count;
961     1
962     1
963 |   1          /************    internal    procedures   ***************/
964     1
965     1
966     1          instructions : proc;
967     2
968     2          put skip(3) edit('####  Input Instructions  ####')(x(20),A);
969     2          put skip(2);
970     2          put skip edit('The input of breakdown descriptions consists of two segments.')(x(10),A);
971     2          put skip edit('In the first segment you are asked to enter three items :   ')(x(5),A);
972     2          put skip edit('(1) the element name, (2) the frequency distributrion name, and')(x(5),A);
973     2          put skip edit('(3) the duration distribution name.  The prompt for both segments')(X(5),A);
974     2          put skip edit('is a "\" .  When this appears on your terminal anter the three') (X(5),A);
975     2          put skip edit('input items separating them by commas. ')(X(5),A);
976     2          put skip(2) edit('You may enter the second segment by answering YES when asked')(X(10),A);
977     2          put skip edit('if there is a breakdown description.  The descriptions may be')(X(5),A);
978     2          put skip edit('up to 5 lines or 400 characters. ')(X(5),A);
979     2
980     2          end instructions;
981     1
982     1
983 |   1          /*****************************************************************************/
984     1
985     1          SCENARIO : Proc;
986     2
987     2
988     2          dcl  string1  char(80) var;
989     2
```

```
  990    2
  991    2
  992    2       dcl line  fixed bin(7) init (0);
  993    2
  994    2       Put skip(2) list('To end the description input type a capital "X" when the prompt appears.');
  995    2
  996    2       put skip(2) list('Enter the description.');
  997    2
  998    2       do while ( line < 5  & string1 ^= 'X' );
  999    3
 1000    3
 1001    3       put skip(2) edit('\',''      '')(A,A);
 1002    3            get edit(string1) (A(80));
 1003    3
 1004    3            string1 = UCASE(string1);
 1005    3          if ( string1 = 'X' ) then do;
 1006    4
 1007    4                                 string1 = '';
 1008    4                                 line = 10;
 1009    4
 1010    4                                 end;
 1011    3
 1012    3
 1013    3            description(count) = description(count) ||  string1;
 1014    3
 1015    3            line =  line + 1;
 1016    3
 1017    3
 1018    3            end;
 1019    2
 1020    2
 1021    2
 1022    2
 1023    2       END SCENARIO;
 1024    1
 1025    1       End DOWN;
```

```
  1  1          Parm_in  : proc ( distribution_name,pointer );
  2  1
  3  1          /*  This procedure sets the parameters for sampling
  4  1              from the given statistical distributions
  5  1
  6  1          */
  7  1
  8  1          %INCLUDE  'UCASE.PLI';
 18  1
 19  1          DCL lex2   entry  (char (80) var,float bin(31),bit(1));
 20  1
 21  1          dcl ERROR  entry  (fixed bin(15) , char (80) var);
 22  1
 23  1          dcl ERRCHK entry  (float bin(31),float bin(31),bit (1));
 24  1
 25  1
 26  1
 27  1          dcl  1   CONSTANT   (100 )   float bin(31)  external;
 28  1
 29  1          dcl  CONS    FIXED BIN(7) EXTERNAL;
 30  1
 31  1
 32  1          dcl  1  EXPONTL ( 100 )     external,
 33  1                  2  mean              float bin(31);
 34  1
 35  1          dcl  1  LOGNORM ( 100 )     external,
 36  1                  2  mean              float bin(31),
 37  1                  2  std               float bin(31);
 38  1
 39  1          dcl  1  NORMAL  ( 100 )     external,
 40  1                  2  mean              float bin(31),
 41  1                  2  std               float bin(31);
 42  1
 43  1          dcl   1   UNIFORM ( 100 )      external,
 44  1                    2  Uppar             float bin(31),
 45  1                    2  Lower             float bin(31);
 46  1
 47  1          dcl   1   TRIANGL ( 100 )      external,
 48  1                    2  Low               float bin(31),
 49  1                    2  Med               float bin(31),
 50  1                    2  High              float bin(31);
 51  1
 52  1
 53  1          dcl   EXPON     fixed bin(7)  external,
 54  1                LOGS      fixed bin(7)  external,
 55  1                NORMS     fixed bin(7)  external,
 56  1                UNIS      fixed bin(7)  external,
 57  1                TRIGS     fixed bin(7)  external;
 58  1
 59  1          dcl   pointer   fixed bin(7);
 60  1
 61  1          dcl distribution_name  char (80) var;
 62  1
 63  1          dcl  error2  bit(1),
```

```
   65     1            err1   bit(1),
   66     1            error1 bit(1),
   67     1
   68     1            true   bit(1) init('1'B),
   69     1            false  bit(1) init('0'B);
   70     1
   71     1
   72     1       dcl  string1  char(80) var,
   73     1            string2  char(80) var;
   74     1
   75     1
   76     1       dcl   value   float bin(31),
   77     1              value1  float bin(31);
   78     1
   79     1
   80  |  1       /*************************************************/
   81     1
   82     1
   83     1       error2 = true;
   84     1
   85     1       do while(error2);
   86     2
   87     2          if( distribution_name = 'EXPONTL' | distribution_name = 'E' )then call P1;
   88     2       else
   89     2
   90     2         if( distribution_name = 'LOGNORM' | distribution_name = 'L' ) then call  P2;
   91     2
   92     2
   93     2       else
   94     2
   95     2         if( distribution_name = 'NORMAL' | distribution_name = 'N' ) then call P3;
   96     2
   97     2       else
   98     2
   99     2         if( distribution_name = 'UNIFORM' | distribution_name = 'U' ) then call P4;
  100     2
  101     2       else if(distribution_name = 'TRIANGL' | distribution_name = 'T') then call  P5 ;
  102     2
  103     2
  104     2       else if( distribution_name = 'CONSTANT' | distribution_name = 'C')then call P6;
  105     2
  106     2          else do;
  107     3
  108     3                put skip(5) list('Invalid distribution  name.');
  109     3                put skip list('Re-enter:');
  110     3                get list(distribution_name);
  111     3               distribution_name = UCASE(distribution_name);
  112     3                error2= true;
  113     3               end;
  114     2
  115     2       END;
  116     1
  117     1
  118     1
  119     1
```

```
122     2
123     2           error2 = false;
124     2           error1 = true;
125     2           do while( error1 );
126     3
127     3            err = true;
128     3            do while ( err );
129     4
130     4               put skip(2) list('Enter the mean of the distribution : ');
131     4               get list(string1);
132     4
133     4                arr = false;
134     4                Call  lex2(string1,value,err );
135     4
136     4                if( err ) then put skip list ('The mean must be numeric. ');
137     4
138     4
139     4                          if( value <= 0.0 ) then do;
140     5                                             Call ERROR(130,string1);
141     5                                             err = '1'B;
142     5                                             end;
143     4
144     4
145     4
146     4                end; /*  inner do while  */
147     3
148     3
149     3                                    error1 = false;
150     3                                    EXPON = EXPON + 1;
151     3                                    EXPONTL.mean(EXPON) = value;
152     3                                    pointer = EXPON;
153     3
154     3
155     3            end; /* end of  error1 do */
156     2
157     2       end   P1;
158     1
159 |   1       /*************************/
160     1
161     1       P2 : proc;
162     2
163     2
164     2
165     2
166     2           error2 = false;
167     2           error1 = true;
168     2           do while ( error1 );
169     3
170     3                  err = true;
171     3                  do while ( err );
172     4
173     4                     put skip(2) list('Enter the mean for the distribution : ');
174     4                     get list(string1);
175     4
 76     4                     err = false;
 77     4                     Call lex2 ( string1, value, err )
173     /
```

```
179     4                       if( err ) then put skip list('The mean must be numeric.');
180     4
181     4                        else do;
182     5                                errorl = false;
183     5                                LOGS = LOGS + 1;
184     5                                pointer = LOGS;
185     5
186     5                                LOGNORM.mean(LOGS) = value;
187     5                                end;
188     4
189     4                     end;
190     3
191     3
192     3           err = true;
193     3
194     3           do while ( err );
195     4
196     4              put skip(2) list('Enter a standard deviation that is greater than 0.0 : ');
197     4
198     4              get list(stringl);
199     4             err = false;
200     4
201     4              Call lex2 ( stringl , value , err );
202     4
203     4               if( err ) then put skip list('The standard deviation must be numeric.');
204     4
205     4              end;
206     3
207     3           if ( value >= 0.0 )then do;
208     4
209     4                        errorl = false;
210     4                        LOGNORM.std( LOGS ) = value;
211     4                         end;
212     3
213     3             else  Call ERROR(101,stringl);
214     3
215     3              end;
216     2
217     2       end P2;
218     1
219     1
220     1       /*************************/
221     1
222     1       P3 : proc;
223     2
224     2
225     2            error2 = false;
226     2            errorl = true;
227     2            do while ( errorl );
228     3
229     3                   err = true;
230     3                   do while ( err );
231     4
232     4                      put skip(2) list('Enter the mean for the distribution : ');
                               get list(stringl);
```

```
                    Call lex2 ( string1, value, err );
                    if( err ) then put skip list('The mean must be numeric.');
                     else do;
                             error1 = false;
                             NORMS = NORMS + 1;
                            pointer = NORMS;
                             NORMAL.mean(NORMS) = value;
                             end;
                  end;

      err = true;
      do while ( err );
         put skip(2) list('Enter a standard deviation that is greater than 0.0 : ');
         get list(string1);
        err = false;
         Call lex2 ( string1 , value , err );
          if( err ) then put skip list('The standard deviation must be numeric.');
         end;
      if ( value >= 0.0 )then do;
                  error1 = false;
                  NORMAL.std( NORMS ) = value;
                   end;
        else  Call ERROR(101,string1);
         end;
   end P3;
   /**********************/

   P4 : proc;

      error2 = false;
      error1 = true;
      do while ( error1 );
       put skip(2) list('Enter the Upper and Lower limits for the Uniform');
       put skip    list('distribution; separating the values by a comma(,): ');
       get list(string1,string2);
```

```
293     3
294     3           err = false;
295     3           Call lex2(string1,value,err);
296     3
297     3           err1 = false;
298     3           Call  lex2(string2,value1,err1);
299     3
300     3               if( err & err1) then do;
301     4
302     4                     put skip(2) list('Both parameters must be numeric => re-enter ');
303     4                     error1 = true;
304     4                     end;
305     3
306     3              else
307     3
308     3                   error1 = false;
309     3
310     3                  end;
311     2
312     2
313     2
314     2
315     2
316     2            if  ( err ) then do;
317     3
318     3              error1 = true;
319     3
320     3              do while(error1 ) ;
321     4
322     4                 put skip(2) list('The Upper limit is invalid => re-enter : ');
323     4                  get list(string1);
324     4                  error1 = false;
325     4                  Call lex2 (string1,value,error1);
326     4
327     4                 if ( ^error1 ) then Call ERRCHK(value1,value,error1);
328     4                 end;
329     3
330     3           UNIS = UNIS + 1;
331     3          pointer = UNIS;
332     3
333     3        UNIFORM.Upper(UNIS) = value;
334     3        UNIFORM.Lower(UNIS) = value1 ;
335     3
336     3           end;
337     2
338     2
339     2        else  if( err1 ) then do;
340     3
341     3           error1 = true;
342     3            do while (error1);
343     4
344     4             put skip(2) list('The  Lower limit is invalid => re-enter : ');
345     4             get list(string2);
346     4
347     4             error1 = false;
348     4             Call  lex2(string2,value1,error1);
```

```
                    end;

                    IF( ^ error1 ) then do;
                                Call  ERRCHK(value1,value,error1);
                                    end;


     UNIS = UNIS + 1 ;
     UNIFORM.Upper(UNIS) = value;
     UNIFORM.Lower(UNIS) = value1 ;
    pointer = UNIS;

               end;  /* end of if do */


   else   if( ^error1 )then do;

          UNIS = UNIS + 1 ;
           pointer = UNIS;

          UNIFORM.Upper(UNIS) = value;
          UNIFORM.Lower(UNIS) = value1;

          end;



end  P4;

/*******************************************************/

P5 : proc;

      error2 = false;
      TRIGS = TRIGS + 1;
      pointer = TRIGS;
      error1 = true;

      do while( error1 );

          put skip(2) list('Enter the Upper and Lower limits for the Triangular distribution:');
          get list(string1,string2);

          Call lex2(string1,value1,err1);
          Call lex2(string2,value,arr);
```

```
407  4
408  4                    do while(err);
409  5
410  5                       put skip(2) list('The lower limit must be numeric => re-enter:');
411  5                       get list(string2);
412  5                       err = false;
413  5                       Call lex2(string2,value,err);
414  5                       end;
415  4
416  4                     error1 = false;
417  4
418  4                     end;
419  3
420  3                else if( ^err & err1 ) then do;
421  4
422  4                    do while( err1 );
423  5
424  5                        put skip(2) list('The upper limit must be numeric => re-enter:');
425  5                        get list(string1);
426  5                        err1 = false;
427  5                        Call lex2(string1,value1,err1);
428  5                        end;
429  4
430  4                     error1 = false;
431  4
432  4                     end;
433  3
434  3
435  3                 else  if( ( err ) & ( err1 ) ) then put skip(2) list('Both limits must be numeric.');
436  3
437  3                 else   error1 = false;
438  3
439  3           end;/* end of do while error1  */
440  2
441  2
442  2           if( value  >=  value1  ) then error1 = true;
443  2
444  2                  do while( error1 );
445  3
446  3                            Put skip(2) list('The Lower limit is greater than or equal the Upper l
447  3
448  3
449  3                     put skip(2) list('Do you wish to change the Upper or Lower limit? ( U/L ):');
450  3                     get list(string1);
451  3                     string1 = UCASE(string1);
452  3
453  3                     if (string1 = 'U' ) then do;
454  4
455  4                           err1 = true;
456  4                           do while( err1 );
457  5
458  5                             put skip list('Enter the new Upper limit:');
459  5                             get list(string1);
460  5                             err1 = false;
461  5                             Call lex2(string1,value1,err1);
462  5                             if(err1) then put skip(2) list('then Upper limit must be-numeric.');
```

```
464     4                                   Call ERRCHK(value,value1,error1);
465     4                                   end;
466     4
467     3
468     3                              else if(string1 = 'L' )then do;
469     4
470     4                                   err1 = true;
471     4                                   do while( err1 );
472     5
473     5                                     put skip list('Enter the new lower limit:');
474     5                                     get list(string2);
475     5                                     err1 = false;
476     5                                     Call lex2(string2,value,err);
477     5
478     5                                     if(err) then put skip(2) list('The Lower limit must be numeric.');
479     5                                     end;
480     4
481     4                                     Call ERRCHK(value,value1,error1);
482     4                                     end;
483     3
484     3                              else  put skip(2) list('Invalid input: Reply( U= Upper , L= Lower)');
485     3
486     3                 end;/*  end of   error1  */
487     2
488     2               TRIANGL.Low(TRIGS) = value;
489     2               TRIANGL.High(TRIGS)= value1;
490     2               error1 = true;
491     2
492     2            do while( error1 );
493     3
494     3               put skip(2) list('Enter the Mode of the distribution: ');
495     3               get list(string1);
496     3               error1 = false;
497     3               Call lex2(string1,value1,error1);
498     3               end;
499     2
500     2          error1 = true;
501     2
502     2
503     2
504     2
505     2               if( value1 > TRIANGL.High(TRIGS) | value1 < TRIANGL.Low(TRIGS))then error1 = true;
506     2
507     2                else  error1 = false;
508     2
509     2
510     2          do while ( error1) ;
511     3
512     3
513     3
514     3                  err1 = true;
515     3                  do while ( err1 );
516     4
517     4                     put skip(2) list('The Mode is out of range.');
518     4                     put skip list('Re-enter:');
519     4                     get list(string1);
```

```
521     4                       Call lex2(string1,value1,err1);
522     4                       end;
523     3
524     3                 if( value1 > TRIANGL.High(TRIGS) | value1 < TRIANGL.Low(TRIGS))then error1 = true;
525     3
526     3                  else  error1 = false;
527     3
528     3               end;
529     2
530     2               TRIANGL.Med(TRIGS) = value1;
531     2
532     2       end  P5;
533     1
534     1
535     1
536 |   1       /***********************************************/
537     1
538     1
539     1       P6 : proc;
540     2
541     2
542     2                     error2 = false;
543     2       CONS = CONS + 1 ;
544     2
545     2       pointer = CONS;
546     2
547     2       error1 = true;
548     2        do while( error1);
549     3
550     3           Put skip(2) list("Enter the value for the constant:  ");
551     3           get edit(string1) ( A(80));
552     3
553     3           error1 = false;
554     3           Call lex2(string1,value,error1);
555     3           if(error1 ) then put skip(2) list("The value must be numeric. ");
556     3
557     3
558     3
559     3             end;
560     2
561     2
562     2               CONSTANT(CONS) = value;
563     2
564     2
565     2       end P6;
566     1
567     1
568     1       end Parm_in;
569
570             OPTIONS : proc ( string );
571     1
572     1
573     1       %INCLUDE  "UCASE.PLI";
583     1
584     1
585     1       dcl e bit(1) init("1"B);
```

```
  587     1       dcl string char(80) var;
  588     1
  589     1       dcl null char(80) var init("");
  590     1
  591     1        do while ( a );
  592     2
  593     2
  594     2       if (string = null)then do;
  595     3                                string = "WC";
  596     3                                a = '0'B;
  597     3                                end;
  598     2
  599     2       else
  600     2          if(string = "L")then do;
  601     3                                a = '0'B;
  602     3                                end;
  603     2
  604     2       else
  605     2          if(string = "UL") then a = '0'B;
  606     2
  607     2       else
  608     2          if(string = "WC") then a ='0'B;
  609     2
  610     2
  611     2       else
  612     2            Call  new_option;
  613     2
  614     2           end;  /* end of do while  */
  615     1
  616     1
  617     1
  618   | 1       /*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
  619     1
  620     1       new_option:  proc;
  621     2
  622     2         put skip list("You entered an invalid work center type.");
  623     2         put skip list("Please  re-enter  ");
  624     2           get edit( string ) (A(80));
  625     2            string = UCASE(string);
  626     2
  627     2         end new_option;
  628     1
  629     1
  630     1       end OPTIONS;
  631             Partsys :  proc;
  632     1
  633     1
  634   | 1       /*********************************************/
  635     1
  636     1
  637     1       %INCLUDE "PARTS.PLI";
  653     1
  654     1
  655     1       dcl        1        Schedule   (25)   external,
  656     1                         2  part_name      char(30) var,
  657     1                         2   lot_size       fixed bin(31),
```

```
659     1                          2   parm_p          fixed bin(7);
660     1
661     1       dcl   NLOTS             fixed bin(7) external;
662     1
663     1
664     1
665 |   1       /*****************************************************************/
666     1
667     1       %include  "UCASE.PLI";
677     1
678     1       dcl   INITPART   entry;
679     1
680     1
681     1       dcl  lex2  entry (char(80) var,float bin(31),bit(1));
682     1            dcl   Parm_in  entry  (char(80) var,fixed bin(7));
683     1       dcl  Rule  entry  (char(80) var, fixed bin(7));
684     1
685     1       dcl   adpart     entry;
686     1
687     1
688     1
689 |   1       /*****************************************************************/
690     1
691     1       dcl ( LL,SETS,rulenum,pointer,J ) fixed bin(7);
692     1       dcl END  BIT(1);
693     1
694     1
695     1       dcl (string0,string1,string2,string3,string4,string5,string6,string7,string8)
696     1           char(80)  var;
697     1
698     1       dcl (err,error1,error2,error3)  bit(1),
699     1           duplicate  bit(1),
700     1           true  bit(1)  init('1'B),
701     1           false  bit(1)  init('0'B);
702     1
703     1       dcl   new bit(1),
704     1             old  bit(1);
705     1
706     1       dcl  (value,value1,value2,value3)  float bin(31);
707     1
708 |   1       /*****************************************************************/
709     1
710     1       call INITPART;
711     1
712     1
713     1       put skip(25);
714     1       put skip(5) edit('####  Part Dascriptions  ####')(X(25),A);
715     1                PARTS = 0;
716     1                PUT SKIP(5) edit ('This is the beginning of the part input sequence')(X(2),A);
717     1                put skip(3) edit('Entar a part name or "QUIT" :  ')(X(10),A);
718     1                GET edit(string1)(A(80));
719     1                string1 = UCASE(string1);
720     1
721     1                     do while ( string1 ^= 'QUIT' );
722     2
723     2                          PARTS = PARTS + 1 ;
```

```
                    error1 = true;
                    do while( error1);
                      put skip(2) edit('How many operations are there for this part?')(X(10),A,X(2));
                      get edit(string2)(A(80));
                      error1 =false;
                      Call lex2( string2,value, error1);
                       if( error1 )then put skip edit('The number of operations is invalid !')(X(5),A);

                     if( value <= 0.0 ) then do;
                            put skip list('The number of operations must be greater than zero !');
                            error1 = true;

                              end;
                       and;/*  end of operations input loop */

                  N_Operations(PARTS) = CEIL(value);

                put skip(5) edit('Enter the distribution name for processing time associated with each opera
                put skip edit('and the part dispatching rule.  The input format is as follows: ')(A);
                put skip(2) edit('Operation 1      distribution_name,dispatching_rule')(A);

                 do  J = 1 to N_Operations(PARTS)  by 1;
                   put skip(2) edit('Operation ',J,'  ')(A,F(2),A);
                   get list(string3,string4);
                    string3 =  UCASE(string3);
                    string4= UCASE(string4);
                     Call Parm_in (string3,pointer);

                   PIECES(PARTS).Operations.Process_d(J) = string3;
                   PIECES(PARTS).Operations.d_pointer(J) = pointer;

                     Call Rule( string4, rulenum);
                    PIECES(PARTS).Operations.Disp_rule(J) = rulenum;


        error2 = true ;
        do while( error2 );
            put skip(5) edit('Next you will enter from 1 to 10 work center names')(X(3),A);
            put skip edit('where operation',J,'can be performed.')(X(3),A,F(2),X(1),A);
            put skip edit('The prompt for input is ">" and you can type')(X(3),A);
            put skip edit(' "STOP" to terminate the input sequence.')(X(2),A);
              put skip(2) edit(' >')(A,X(3));
              get edit (string5) (A(30));
              string5 = UCASE(string5);
              SETS = 0 ;

              DO while( string5 ^= 'STOP' &  SETS < 10 );

                 SETS = SETS + 1 ;
                 PIECES(PARTS).Operations(J).Centers(SETS) = string5;
                 put skip(2) edit(' >')(A,X(3));
                 get edit(string5)(A(80));
                 string5 = UCASE(string5);
                 end;
```

```
                                   put skip list('You must enter at least one work center name.');
                                   error2 = true;
                                   end;
                          else   error2 = false;
          END;



        end; /*  end of individual  operations input  */
      /*************************************************/
        PUT skip(5) edit('The next input section involves specifying the sequencing')(X(2),A);
      put skip edit('of operations.  When you are given  an "*" as a prompt you ')(X(2),A);
      put skip edit('can enter the operation numbers in the order that they')(X(2),A);
      put skip edit('are to be performed similar to the following example :  ')(X(2),A);
        put skip(2) edit('Assume there are 5 different operations, the input sequence')(X(3),A);
        put skip edit('would look like the following :')(X(3),A);
        put skip(2) edit('Operation Sequence    ')(X(5),A);
        put skip edit('*    4')(X(5),A);
        put skip edit('*    1')(X(5),A);
        put skip edit('*    5')(X(5),A);
        put skip edit('*    2')(X(5),A);
        put skip edit('*    3')(X(5),A);

             END = false ;
                      put skip(3) edit('Operation Sequencing')(X(5),A);

               do  LL = 1 to PIECES.N_Operations(PARTS)  by 1;

                 error3 = true;
                 do while ( error3 );
                   put skip(2) edit('*')(A,X(3));
                   get edit ( string6)(A(80));
                   string6 = UCASE(string6);
                    error3 = false;
                     Call lex2(string6,value1,error3);
                     if(error3)then put skip list('The input value must be numeric.');
                     end;

              PIECES(PARTS).Q_numbers(LL) = CEIL(value1);
             end;


         put skip(3) edit('Enter a part name or "QUIT"  :  ')(X(2),A);
         get edit(string1)(A(80));
          string1 = UCASE(string1);
      END;  /* end of new loop */


      Put skip(3) edit('Would you like to make any changes? ( Y/N ):  ')(X(20),A);
              get edit(string1) ( A(90));
              string1 = UCASE(string1);
              if( string1 = 'Y')then call adpart;
```

```
839  1          put skip(5) edit('This is the beginning of the Production Schedule input sequence.')(X(5),A);
840  1        put skip(2) edit('In this section you will be asked to input three items')(X(5),A);
841  1        put skip edit('to describe the production schedule.  The three items are the')(X(5),A);
842  1        put skip edit('part name, the lot size, and the frequency for loading individual')(X(5),A);
843  1      put skip edit('parts into the system.  The production lots should be entered in the')(X(5),A);
844  1    put skip edit('sequence you would like them to be processed.')(X(5),A);
845  1
846  1    NLOTS = 0 ;
847  1
848  1    do while( NLOTS < 1 );
849  2
850  2       put skip(2) edit('Enter a part name or "QUIT" :  ')(X(5),A);
851  2       get edit(string1)(A(80));
852  2       string1 = UCASE(string1);
853  2       do while( string1 ^= 'QUIT');
854  3
855  3           duplicate = false;
856  3           do while ( ^duplicate );
857  4
858  4                do LL = 1 to PARTS by 1;
859  5
860  5                   if ( string1 = P_name(LL))then duplicate = true;
861  5                   end;
862  4
863  4                           if( string1 = 'QUIT')then goto  OUT;
864  4
865  4
866  4                if( ^ duplicate ) then do;
867  5
868  5                    put skip(2) edit('The part ',string1,' does not exist!')(X(2),A,A,A);
869  5                  Put skip(3) edit('The following are valid part names.')(X(5),A);
870  5
871  5                do LL = 1 to PARTS by 2;
872  6
873  6                  put skip(2) edit(PIECES.P_name(LL), PIECES.P_name(LL+1))(X(5),A(30),X(5),A(30));
874  6                        end;
875  5                             put skip edit('Enter a new part name : ')(X(5),A);
876  5                             get edit(string1)(A(80));
877  5                             string1 = UCASE(string1);
878  5                             duplicate = false;
879  5                             end;
880  4
881  4
882  4
883  4                       end;
884  3
885  3                put skip(4) edit('Enter the "lot size" and the "input frequency"')(X(5),A);
886  3                put skip edit('seperating the items by a comma.')(X(5),A);
887  3                put skip edit(':  ')(X(5),A);
888  3                get list(string2,string3);
889  3                string3 = UCASE(string3);
890  3
891  3                call lex2(string2,value,err);
892  3
893  3                do while( value <= 0.0 );
994  4
```

```
                        put skip edit('Re-enter :  ')(X(5),A);
                        get edit(string2)(A(80));
                        call lex2(string2,value,err);
                        end;

                    NLOTS = NLOTS + 1;

                    call Parm_in(string3,parm_p(NLOTS));


                    Part_name(NLOTS) = string1;
                    lot_size(NLOTS)  = CEIL(value);
                    Freq_d(NLOTS)    = string3;

               put skip(2) edit('Enter a part name or "QUIT" :    ')(X(5),A);
               get edit(string1)(A(80));
               string1 = UCASE(string1);
               OUT:       end;

     if(NLOTS < 1 ) then put skip(2) edit('You must enter at least 1 production lot !')(X(2),A);

     end;


     End Partsys;
     Rule :   proc ( name, p );

     dcl name      char(90) var,
         p         fixed bin(7),
         error1    bit(1),
         true      bit(1) init('1'B),
         false     bit(1) init('0'B);

     dcl  ERROR  entry( fixed bin(15), char(80) var);

     %include  'UCASE.PLI';

     /*********************************************/

     error1 = true;

     do while ( error1 );

          if(name = 'SNIQ')then do;

                        p = 1;
                        error1 = false;
                        end;

     else if(name = 'MHIT')then do;

                        p = 2;
                        error1 = false;
                        end;
```

```
 962     2      else if(name = 'RAND')then do;
 963     3
 964     3                          p = 3;
 965     3                          error1 = false;
 966     3                          end;
 967     2
 968     2      else if(name = 'LAPT')then do;
 969     3
 970     3                          p = 4;
 971     3                          error1 = false;
 972     3                          end;
 973     2
 974     2      else do;
 975     3
 976     3             call ERROR(131,name);
 977     3
 978     3              put skip(2) list('The following are valid dispatching rules: ');
 979     3              put skip(2) list('SNIQ  ==>  Smallest number in the queue.');
 980     3              put skip    list('MMIT  ==>  Machine with the highest idle time.');
 981     3               put skip   list('RAND  ==>  Select a machine at random.');
 982     3               put skip   list('LAPT  ==>  Select the machine with the least');
 983     3               put skip   list('                amount of processing time in the queue.');
 984     3
 985     3             put skip(2) list('Enter a new dispatching rule :  ');
 986     3             get edit(name) (A(80));
 987     3             name = UCASE(name);
 988     3             error1 = true;
 989     3             end;
 990     2
 991     2      end;
 992     1
 993     1
 994     1      end Rule;
 995
 996 |          /*********************************************************/
 997
 998          INITPART : proc;
 999     1
1000     1
1001     1      %include 'PARTS.PLI';
1017     1
1018     1
1019     1
1020     1
1021     1      dcl (I,J,K)  fixed bin(7);
1022     1
1023     1      do I = 1 to 20;
1024     2
1025     2         P_name(I) = '';
1026     2         do J = 1 to 50 ;
1027     3
1028     3             PIECES(I).Operations(J).Process_d = '';
1029     3             PIECES(I).Operations(J).d_pointer = 0;
1030     3             PIECES(I).Operations(J).Disp_rule = '';
1031     3             do  K = 1 to 10;
1032     4
```

```
1034    4                 end;
1035    3
1036    3              end;
1037    2
1038    2
1039    2                 do J = 1 to 50;
1040    3
1041    3                    PIECES(I).D_numbers(J) = 0;
1042    3                    end;
1043    2
1044    2          end;
1045    1
1046    1       end INITPART;
1047            adpart : proc;
1048    1
1049    1
1050    1       DCL      reply              char(80) var,   err              bit(1),
1051    1                false    bit(1) init('0'B),      true     bit(1) init('1'B),
1052    1                Mod_name           entry,
1053    1                Add_op             entry,
1054    1                delete_op          entry,
1055    1                Mod_op             entry,
1056    1                New_rout           entry;
1057    1
1058    1
1059    1       do while( reply ^= '6');
1060    2
1061    2                put skip(25) edit('Part Editor')(X(20),A);
1062    2                put skip(2) edit('Option          Function')(X(20),A);
1063    2                put skip(2) edit('   1        Change a part name')(X(20),A);
1064    2                put skip(2) edit('   2        Add an operation')(X(20),A);
1065    2                put skip(2) edit('   3        Delete an operation')(X(20),A);
1066    2                put skip(2) edit('   4        Modify an opeation')(X(20),A);
1067    2                put skip(2) edit('   5        Enter a new operation sequence')(X(20),A);
1068    2                put skip(2) edit('   6        Exit')(X(20),A);
1069    2                put skip(2) edit('Enter the funciton you would like to perform :  ')(X(20),A);
1070    2                get edit(reply)(A(80));
1071    2                arr = true;
1072    2                do while( arr );
1073    3
1074    3                        if( reply = '1')then do; arr = false;
1075    4                                                 call Mod_name;
1076    4                                                 end;
1077    3
1078    3                        else if( reply = '2')then do; arr = false;
1079    4                                                      call Add_op;
1080    4                                                      end;
1081    3
1082    3                        else if( reply = '3')then do;  arr = false;
1083    4                                                      call delete_op;
1084    4                                                        end;
1085    3
1086    3                        else if( reply = '4')then do;  arr = false;
1087    4                                                      call Mod_op;
1088    4                                                      end;
1089    3
```

```
1091    4                                                       call New_rout;
1092    4                                                       end;
1093    3
1094    3                       else if( reply = "5")then err = false;
1095    3
1096    3                               else do;
1097    4
1098    4                               put skip(2) edit("Invalid input, re-anter :  ")(X(10),A);
1099    4                               get edit(reply)(A(80));
1100    4                               end;
1101    3
1102    3               end;
1103    2
1104    2       end;
1105    1
1106    1       end edpart;
1107
1108    1       /*************************************************/
1109
1110            Mod_name : proc;
1111    1
1112    1       %include "PARTS.PLI";
1128    1
1129    1       %INCLUDE "UCASE.PLI";
1139    1
1140    1       dcl     found           bit(1),         name            char(80)var,
1141    1               J               fixed bin(7),   true    bit(1) init("1"B),
1142    1
1143    1       Print_parts     entry;
1144    1
1145    1
1146    1
1147    1       found = "0"B;
1148    1       do while( ^found );
1149    2
1150    2               put skip(25) edit("Enter the part name you would like to change : ")(X(5),A);
1151    2               get edit( name)(A(80));
1152    2               name = UCASE(name);
1153    2               J = 0;
1154    2               do while( ^found & J < PARTS );
1155    3
1156    3                       J = J + 1;
1157    3                       if( P_name(J) = name)then found = true;
1158    3                       end;
1159    2
1160    2               if( ^found)then do;
1161    3
1162    3                       put skip(3) edit("Invalid part name!","The following are valid part names.")(X(5),A,skip
1163    3                       call Print_parts;
1164    3                       end;
1165    2
1166    2               end;
1167    1
1168    1
1169    1       P_name(J) = name;
1170    1
```

```
1172
1173 |         /*********************************************************/
1174
1175            Add_op  : proc;
1176     1
1177     1      %INCLUDE "UCASE.PLI";
1187     1
1188     1      %INCLUDE "PARTS.PLI";
1204     1
1205     1      dcl
1206     1              reply           char(80) var,   J               fixed bin(7),
1207     1              string1         char(80) var,   string2         char(80) var,
1208     1              pointer         fixed bin(7),   rulenum         fixed bin(7),
1209     1              place           fixed bin(7),   error2          bit(1),
1210     1
1211     1              true    bit(1) init('1'B),      false   bit(1) init('0'B),
1212     1              SETS            fixed bin(7),   string5         char(80) var;
1213     1
1214     1      dcl
1215     1
1216     1      Print_parts     entry,
1217     1      Psearch         entry   (char(30) var,fixed bin(7)),
1218     1      Rule            entry   (char(30) var,fixed bin(7)),
1219     1      Parm_in         entry   (char(30) var,fixed bin(7));
1220     1
1221     1
1222     1
1223     1      call Print_parts;
1224     1
1225     1      put skip edit('Enter a part to add an operation to or "QUIT" :  ')(X(5),A);
1226     1      get edit(reply)(A(80));
1227     1      reply = UCASE(reply);
1228     1      do while( reply ^= 'QUIT');
1229     2
1230     2              call Psearch(reply, J );
1231     2              do while( J < 1);
1232     3
1233     3                      put skip(2) edit('Invalid part name, re-enter :  ')(X(5),A);
1234     3                      get edit(reply)(A(80));
1235     3                      reply = UCASE(reply );
1236     3                      call Psearch(reply , J );
1237     3                      end;
1238     2
1239     2              if( PIECES.N_Operations(J) > 49 ) then do;
1240     3
1241     3                      put skip edit('There are already 50 operations.',
1242     3                      'You cannot add an operation.')(
1243     3                      X(5),A,SKIP,X(5),A);
1244     3                      end;
1245     2
1246     2              else do;
1247     3
1248     3                              PIECES.N_Operations(J) = N_Operations(J) + 1;
1249     3                      place = N_Operations(J);
1250     3                      put skip(2) edit('Enter the processing distribution and dispatch rule,',
1251     3                                      'seperated by a comma.')(X(5),A,
```

```
 1253     3
 1254     3                              get list(string1,string2);
 1255     3                         string1 = UCASE(string1);
 1256     3                         string2 = UCASE(string2);
 1257     3                         call Parm_in(string1,pointer);
 1258     3
 1259     3                         PIECES(J).Operations.Process_d(place) = string1;
 1260     3                         PIECES(J).Operations.d_pointer(place) = pointer;
 1261     3
 1262     3                         call Rule(string2,rulenum);
 1263     3                         PIECES(J).Operations.Disp_rule(place) = rulenum;
 1264     3
 1265     3           error2 = true ;
 1266     3           do while( error2 );
 1267     4               put skip(5) edit('Next you will enter from 1 to 10 work center names')(X(3),A);
 1268     4               put skip edit('where operation',J,'can be performed.')(X(3),A,F(2),X(1),A);
 1269     4               put skip edit('The prompt for input is ">" and you can type')(X(3),A);
 1270     4               put skip edit(' "STOP" to terminate the input sequence.')(X(2),A);
 1271     4                 put skip(3) edit(' >')(A,X(3));
 1272     4                 get edit (string5) (A(80));
 1273     4                 string5 = UCASE(string5);
 1274     4                 SETS = 0 ;
 1275     4
 1276     4                 DO while( string5 ^= 'STOP' &  SETS < 10 );
 1277     5
 1278     5                    SETS = SETS + 1 ;
 1279     5                    PIECES(PARTS).Operations(J).Centers(SETS) = string5;
 1280     5                    put skip(2) edit(' >')(A,X(3));
 1281     5                    get edit(string5)(A(80));
 1282     5                    string5 = UCASE(string5);
 1283     5                    end;
 1284     4
 1285     4                    if( SETS = 0 )THEN do;
 1286     5                              put skip list('You must enter at least one work center name.');
 1287     5                              error2 = true;
 1288     5                              end;
 1289     4                    else    error2 = false;
 1290     4              END;
 1291     3
 1292     3
 1293     3        end;
 1294     2
 1295     2        put skip(2) edit('Enter a part to add an operation to or "QUIT"  :  ')(X(5),A);
 1296     2        get edit(reply)(A(80));
 1297     2        reply = UCASE(reply);
 1298     2        end;
 1299     1
 1300     1        end Add_op;
 1301
 1302
 1303 |           /*****************************************************************************/
 1304
 1305             delete_op : proc;
 1306     1
 1307     1        %include 'PARTS.PLI';
 1323     1        %INCLUDE 'UCASE.PLI';
```

```
1334    1       dcl
1335    1               reply           char(80) var,   J               fixed bin(7),
1336    1               err             bit(1),         string1         char(80)var,
1337    1               value           float bin(31),  the_one         fixed bin(7),
1338    1               K               fixed bin(7),
1339    1
1340    1       lex2    entry   (char(80) var,float bin(31),bit(1)),
1341    1       Psearch entry   (char(80) var,fixed bin(7)),
1342    1       Print_parts     entry;
1343    1
1344    1
1345    1
1346    1
1347    1       Call Print_parts;
1348    1
1349    1       put skip(2) edit('Enter a part to delete of "QUIT"  : ')(X(5),A);
1350    1       get edit(reply)(A(80));
1351    1       reply = UCASE(reply);
1352    1       do while( reply ^= 'QUIT');
1353    2
1354    2               call Psearch(reply, J );
1355    2               do while( J < 1 );
1356    3
1357    3                       put skip(2) edit('Invalid part name, re-enter :  ')(X(5),A);
1358    3                       get edit(reply)(A(80));
1359    3                       reply = UCASE(reply );
1360    3                       call Psearch(reply , J );
1361    3                       end;
1362    2
1363    2
1364    2               err = '1'B;
1365    2               do while( err );
1366    3
1367    3                       put skip(2) edit('Enter the operation number to delete :  ')(X(5),A);
1368    3                       get edit(string1) (A(80));
1369    3                       call lex2(string1,value,err );
1370    3                       if( err ) then put skip edit('Invalid operation number.')(X(10),A);
1371    3                       end;
1372    2
1373    2               the_one = CEIL(value);
1374    2
1375    2               do K = the_one  to  (N_Operations(J) -1 );
1376    3
1377    3                       PIECES(J).Operations(K) = PIECES(J).Operations(K+ 1);
1378    3                       end;
1379    2
1380    2               N_Operations(J) = N_Operations(J) - 1;
1381    2
1382    2       put skip(2) edit('Enter a part to delete of "QUIT"  : ')(X(5),A);
1383    2       get edit(reply)(A(80));
1384    2       reply = UCASE(reply);
1385    2
1386    2       END;
1387    1
1388    1
1389    1       end delete_op;
```

```
1391
1392 1          /*****************************************************************************/
1393
1394            New_rout : proc;
1395    1
1396    1       %include 'UCASE.PLI';
1406    1       %INCLUDE 'PARTS.PLI';
1422    1
1423    1
1424    1
1425    1       dcl
1426    1               LL      fixed bin(7),   END     bit(1), false   bit(1) init('0'B),
1427    1               rulenum         fixed bin(7),
1428    1
1429    1               error3  bit(1), true    bit(1) init('1'B),
1430    1               string6 char(80) var,   value1  float bin(31),
1431    1
1432    1       Psearch         entry   (char(80) var,fixed bin(7)),
1433    1       Print_parts     entry,
1434    1       lex2    entry   (char(80) var,float bin(31), bit(1));
1435    1
1436    1
1437    1       LL = 0;
1438    1       do while ( LL < 1);
1439    2
1440    2               put skip(5) edit('Enter a part name : ')(X(10),A);
1441    2               get edit(string6)(A(80));
1442    2               string6 = UCASE(string6);
1443    2               call Psearch(string6,LL);
1444    2               if( LL < 1 ) then do;
1445    3                       put skip edit('The following are valid part names.')(X(5),A);
1446    3                       call Print_parts;
1447    3                       end;
1448    2
1449    2               end;
1450    1
1451    1       rulenum = LL;
1452    1
1453    1
1454    1         PUT skip(5) edit('This  section involves specifying the sequencing')(X(2),A);
1455    1       put skip edit('of operations.  When you are given  an "*" as a prompt you ')(X(2),A);
1456    1       put skip edit('can enter the operation numbers in the order that they')(X(2),A);
1457    1       put skip edit('are to be performed similar to the following example :  ')(X(2),A);
1458    1         put skip(2) edit('Assume there are 5 different operations, the input sequence')(X(3),A);
1459    1         put skip edit('would look like the following :')(X(3),A);
1460    1         put skip(2) edit('Operation Sequence    ')(X(5),A);
1461    1         put skip edit('*    4')(X(5),A);
1462    1         put skip edit('*    1')(X(5),A);
1463    1         put skip edit('*    5')(X(5),A);
1464    1         put skip edit('*    2')(X(5),A);
1465    1         put skip edit('*    3')(X(5),A);
1466    1
1467    1              END = false :
1468    1                       put skip(3) edit('Operation Sequencing')(X(5),A);
1469    1
1470    1                do  LL = 1 to PIECES.N_Operations(rulenum)  by 1;
1471    2
```

```
1472     2                  error3 = true;
1473     2                  do while ( error3 );
1474     3                    put skip(2) edit('*')(A,X(3));
1475     3                    get edit ( string6)(A(80));
1476     3                    string6 = UCASE(string6);
1477     3                     error3 = false;
1478     3                      Call lex2(string6,value1,error3);
1479     3                      if(error3)then put skip list('The input value must be numeric.');
1480     3                       end;
1481     2
1482     2               PIECES(rulenum).D_numbers(LL) = CEIL(value1);
1483     2              end;
1484     1
1485     1
1486     1       end New_rout;
1487
1488 |           /*****************************************************************************/
1489
1490             Mod_op : proc;
1491     1
1492     1       %INCLUDE 'UCASE.PLI';
1502     1       %INCLUDE 'PARTS.PLI';
1518     1
1519     1
1520     1
1521     1       dcl
1522     1               reply           char(80) var,   J               fixed bin(7),
1523     1               string1         char(80) var,   err             bit(1),
1524     1                       opnum                   fixed bin(7),   value           float bin(31),
1525     1               option          char(80) var,   location        fixed bin(7),
1526     1                       point                   fixed bin(7),   rulenum         fixed  bin(7),
1527     1
1528     1       Parm_in         entry   (char(30) var,fixed bin(7)),
1529     1       Rule            entry   (char(80) var,fixed bin(7)),
1530     1       lex2            entry   (char(30) var,float bin(31),bit(1));
1531     1
1532     1
1533     1
1534     1       call Print_parts;
1535     1       put skip edit('Enter a part name to change or "QUIT" :  ')(X(5),A);
1536     1       get edit(reply)(A(80));
1537     1       reply = UCASE(reply);
1538     1       do while( reply ^= 'QUIT');
1539     2
1540     2               call Psearch(reply,J);
1541     2               do while( J < 1 );
1542     3
1543     3
1544     3                       put skip(2) edit('Invalid part name, re-enter :  ')(X(5),A);
1545     3                       get edit(reply)(A(80));
1546     3                       reply = UCASE(reply );
1547     3                       call Psearch(reply , J );
1548     3                       end;
1549     2
1550     2               err = '1'B;
1551     2               do while( err );
1552     3
```

```
1553    3                       put skip(3) edit('Which operation do you wish to modify :  ')(X(5),A);
1554    3                       get edit(string1)(A(80));
1555    3                       call lex2(string1,value,err);
1556    3                       if( err )then put skip edit('Invalid operation number !')(X(5),A);
1557    3                       end;
1558    2
1559    2               opnum = CEIL(value);
1560    2
1561    2               put skip(2) edit('Do you wish to change,',
1562    2                                'Processing distribution(D),',
1563    2                                'Processing parameters(P),',
1564    2                                'Dispatching rule(R),',
1565    2                                'Work Centers(W)    : ')(X(10),A,SKIP,
1566    2                                                         X(10),A,SKIP,
1567    2                                                         X(10),A,SKIP,
1568    2                                                         X(10),A,SKIP,
1569    2                                                         X(10),A);
1570    2
1571    2               err = '1'B;
1572    2               do while( err ) ;
1573    3
1574    3                       get edit(option)(A(80));
1575    3                       option  = UCASE(option);
1576    3                       if( option = 'D' | option = 'P' |
1577    3                           option = 'R' | option = 'W' )then err = '0'B;
1578    3
1579    3               else put skip edit('Invalid option, reenter')(X(10),A);
1580    3
1581    3               end;
1582    2
1583    2       if( option = 'D' )then do;
1584    3
1585    3               put skip(2) edit('Enter the new distribution name : ')(X(10),A);
1586    3               get edit(string1)(A(30));
1587    3               string1 = UCASE(string1);
1588    3               call Parm_in(string1,location);
1589    3               PIECES(point).Operations(opnum).Process_d = string1;
1590    3               PIECES(point).Operations(opnum).d_pointer = location;
1591    3               end;
1592    2
1593    2       else if( option = 'P')then do;
1594    3
1595    3               call Parm_in(PIECES(point).Operations(opnum).Process_d,location);
1596    3               PIECES(point).Operations(opnum).d_pointer = location;
1597    3               end;
1598    2
1599    2       else if( option = 'R')then do;
1600    3
1601    3               put skip(2) edit('Enter the new dispatching rule :  ')(X(5),A);
1602    3               get edit(string1)(A(80));
1603    3               string1 = UCASE(string1);
1604    3               call Rule(string1,rulenum);
1605    3               PIECES(point).Operations(opnum).Disp_rule = rulenum;
1606    3               end;
1607    2
1608    2
```

```
1610     3                 put skip(2) edit('Enter the new work center names when you get a ">"')(X(5),A);
1611     3                 put skip edit('Enter "STOP" and the input.')(X(5),A);
1612     3                 J = 0;
1613     3                 do while( J < 10  & string1 ^= 'STOP' );
1614     4
1615     4                         put skip(2) edit(">")(X(5),A);
1616     4                         get edit(string1)(a(80));
1617     4                         string1 = UCASE(string1);
1618     4                         if( string1 = 'STOP' & J = 0) then
1619     4
1620     4                                 put skip edit('You must enter atleast one work center.')(X(5),A);
1621     4                         else do;
1622     5
1623     5                         J = J + 1;
1624     5                         PIECES(point).Operations(opnum).Centers(J) = string1;
1625     5                         end;
1626     4                 END;
1627     3
1628     3        END;
1629     2
1630     2
1631     2        put skip edit('Enter a part name to change or "QUIT" :  ')(X(5),A);
1632     2        get edit(reply)(A(80));
1633     2        reply = UCASE(reply);
1634     2
1635     2        end;
1636     1
1637     1
1638     1        end     Mod_op;
1639
1640
1641 |           /*****************************************************************************/
1642
1643
1644              Print_parts : proc;
1645     1
1646     1        dcl     J       fixed bin(7);
1647     1        %include 'PARTS.PLI';
1648     1
1664     1        do J = 1 to PARTS by 2;
1665     2
1666     2                 put skip edit(P_name(J),P_name(J+1))(X(5),A(30),X(5),A(30));
1667     2                 end;
1668     1
1669     1        end Print_parts;
1670
1671
1672 |           /*************************************************/
1673
1674              Psearch : proc(name,J);
1675     1
1676     1        %include 'PARTS.PLI';
1677     1        dcl  name        char(80) var,   J   fixed bin(7),  K  fixed bin(7),
1693     1             found       bit(1);
1694     1
1695     1
```

```
1697     1      found = '0'B;
1698     1      do while( ^found & J < PARTS );
1699     2
1700     2              K = K + 1;
1701     2              if( P_name(K) = name ) then found = '1'B;
1702     2              end;
1703     1
1704     1      if( found ) then J = K ;
1705     1      else J = 0;
1706     1
1707     1      end Psearch;
1708
```

```
)     2
)     2       lower = lower + 20;
      2       test = upper + 20;
      2       upper = MIN(test,NUMBRKS);
      2       end;
      1
      1
      1       ans = '0'B;
      1       do while( ^ans);
      2
      2               put skip edit('Would you like this report saved ? ')(X(10),A);
      2               put skip edit('Reply( YES/NO ): ')(X(10),A);
      2               get edit(string)(A(80));
      2               string = UCASE(string);
      2               if( string = 'YES' | string = 'NO' ) then ans = '1'B;
      2               else put skip edit('Invalid reply')(X(10),A);
      2               end;
      1
      1       if( string = 'YES')then call DOWNHARD;
      1
      1
      1       heading : proc;
      2
      2       put skip(25) edit('Breakdown Summary Report')(X(24),A);
      2       put skip(2) edit('Breakdown','Element','Average','Frequency')(A,X(10),A,X(8),A,X(5),A);
      2       put skip edit('Number','Downtime','per unit time')(A,X(29),A,X(4),A);
      2
      2       end heading;
      1
      1       end DOWNSTAT;
```

```
     1
     1        heading2 : proc;
     2
     2        PUT FILE (WIPSTATS) PAGE SKIP edit('Production by Part Type')(X(24),A);
     2        PUT FILE(WIPSTATS) SKIP(2) EDIT('MEAN','MEAN')(X(34),A(4),X(18),A(4));
     2        PUT FILE(WIPSTATS) SKIP EDIT('PRODUCTION','TIME IN')(X(31),A(10),X(13),A(7));
     2        PUT FILE(WIPSTATS) SKIP EDIT('RATE','VARIANCE','SYSTEM','VARIANCE')
     2                        (X(34),A(4),X(5),A(8),X(4),A(6),X(4),A(8));
     2
     2        end heading2;
     1
     1        end WIPHARD;

              /*******************************************************/

              DOWNSTAT : PROC;
     1
     1        %INCLUDE 'MACHINE.PLI';
     1        %INCLUDE 'BREAKS.PLI';
     1        %include 'UCASE.PLI';
     1
     1        dcl   (pages,lower,upper,test,K,J     )     fixed bin(7);
     1        dcl  (m1,m2,m3,v1,v2,v3              )    float bin(31);
     1
     1        dcl  ans   bit(1);
     1        dcl  string   char(80) var;
     1
     1
     1        dcl  DOWNHARD   ENTRY;
     1
     1        DCL  Calc_stat  entry(fixed bin(7),fixed bin(7),float bin(31),
     1                              float bin(31),float bin(31),float bin(31),
     1                              float bin(31),float bin(31));
     1
     1
     1        if( NUMBRKS <= 20 ) then pages = 1;
     1        else if( NUMBRKS > 20  & NUMBRKS <= 40 )then pages = 2;
     1        else if( NUMBRKS > 40  & NUMBRKS <= 60 )then pages = 3;
     1        else if( NUMBRKS > 60  & NUMBRKS <= 80 )then pages = 4;
     1        else pages = 5;
     1
     1        lower = 1;
     1        upper = MIN(20,NUMBRKS);
     1
     1        do J = 1 to pages;
     2
     2        call heading;
     2
     2                do K = lower to upper;
     3
     3                        call Calc_stat(7,K,m1,v1,m2,v2,m3,v3);
     3                        put skip edit(K,Element(K),m1,v1)
     3                        (x(3),F(3),x(5),A(30),x(5),F(7,3),x(5),f(7,3));
     3                        end;
     2
     2                put skip edit('Hit a carriage return to continue.')(X(5),A);
     2                get edit(string)(A(80));
```

```
1
1       END MHARD;
        WIPHARD : proc;
1
1       %include 'PARTS.PLI';
1       %INCLUDE 'UCASE.PLI';
1
1       DCL   WIPSTATS   FILE STREAM OUTPUT PRINT;
1
1
1       DCL ( m1,v1,m2,v2,m3,v3  ) float bin(31);
1       dcl  reply                 char(80) var;
1       dcl  J                     fixed bin(7);
1       DCL   err  bit(1);
1
1
1       OPEN FILE(WIPSTATS);
1
1
1       dcl  Calc_stat   entry ( fixed bin(7),fixed bin(7),float bin(31),
1                                float bin(31),float bin(31),float bin(31),
1                                float bin(31),float bin(31));
1
1
1       call heading1;
1       PUT FILE (WIPSTATS) skip(3) edit('Work in Process')(X(28),A);
1       call Calc_stat(4,0,m1,v1,m2,v2,m3,v3);
1       PUT FILE (WIPSTATS) skip(2) edit('Average number of parts in the system =  ',m1)(X(12),A,F(6,2));
1       PUT FILE (WIPSTATS) skip(2) edit('Variance                              =  ',v1)(X(12),A,F(6,2));
1       CALL Calc_stat(8,0,m1,v1,m2,v2,m3,v3);
1       PUT FILE (WIPSTATS) skip(4) edit('System Production')(X(28),A);
1       PUT FILE (WIPSTATS) skip(2) edit('Average Time in the System =  ',m1)(X(20),A,F(7,3));
1       PUT FILE (WIPSTATS) skip(2) edit('Variance                   =  ',v1)(X(20),A,F(7,3));
1       IF( m1 > 0.0 ) then m1  = 1/m1;
1       else m1 = 0.0;
1       if( v1 > 0.0 ) then v1 = 1/v1;
1       else v1 = 0.0;
1
1       PUT FILE (WIPSTATS) skip(2) edit('Part Production per unit time =  ',m1)(X(20),A,F(7,3));
1       PUT FILE (WIPSTATS) skip(2) edit('Variance                      =  ',v2)(X(20),A,F(7,3));
1       call heading2;
1
1       do J = 1 to PARTS;
2
2               call Calc_stat(9,J,m1,v1,m2,v2,m3,v3);
2
2               PUT FILE (WIPSTATS) skip edit(PIECES.P_name(J),m2,v2,m1,v1)
2               (A(30),4(x(4),F(7,3)));
2               end;
1
1
1
1       heading1 : proc;
2
2       PUT FILE (WIPSTATS) PAGE SKIP edit('Work in Process and Production Report')(X(17),A);
2       end heading1;
```

```
1                          float bin(31),float bin(31));
1
1       dcl   (pages,J,K,lower,upper,test) fixed bin(7);
1
1       dcl   (m1,v1,m2,v2,m3,v3,m11,v11,m21,v21,m31,v31 ) float bin(31);
1
1       dcl   no  bit(1);
1       dcl    MAXCARTS   FIXED BIN(15) EXTERNAL;
1
1       dcl   reply  char(80) var;
1       OPEN FILE(MHSTATS);
1
1
1       if( PATHS <= 20 ) THEN pages = 1;
1       else if( PATHS > 20  & PATHS <= 40 ) THEN pages = 2;
1       else if( PATHS > 40  & PATHS <= 60 ) then pages = 3;
1       else if( PATHS > 60  & PATHS <= 80 ) THEN pages = 4;
1       else pages = 5;
1
1       lower = 1;
1       upper = MIN(20,PATHS);
1       do J = 1 to pages;
2
2               call pagehead;
2               do K = lower to upper by 2;
3
3                       call Calc_stat(5,K,m1,v1,m2,v2,m3,v3);
3                       call Calc_stat(5,K+1,m11,v11,m21,v21,m31,v31);
3                       PUT FILE (MHSTATS) skip edit(K,m1,v1,m2,v2,K+1,m11,v11,m21,v21)                    (X(4),F(3),X
3                         F(5,2),X(3),F(7,3),X(5),F(3),X(5),F(5,2),
3                    X(3),F(7,3),X(5),F(5,2),X(3),F(7,3));
3
3               END;
2
2               test = upper + 20;
2               lower = lower + 20;
2               upper = MIN(test,20);
2               end;
1
1       PUT FILE (MHSTATS) PAGE SKIP edit("CARRIER UTILIZATION")(x(25),A);
1       PUT FILE (MHSTATS) skip(2) edit("Number of","Average","Variance")(X(12),A,X(12),A,X(12),A);
1       PUT FILE (MHSTATS) skip edit("Carriers","%Utilization")(X(12),A,X(13),A);
1       call Calc_stat(6,0,m1,v1,m2,v2,m3,v3);
1       PUT FILE (MHSTATS) skip edit(MAXCARTS,m1,v1)(X(14),f(3),x(14),F(6,2),X(14),F(7,3));
1
1
1
1
1       pagehead : proc;
2
2       PUT FILE (MHSTATS) PAGE SKIP edit("Material Handling Utilization Report")(X(18),A);
2       PUT FILE (MHSTATS) skip(2) edit("PATH","MEAN","VARIANCE","PATH","MEAN","VARIANCE")(X(3),A,X(5),A,X(5),A,X(5),A,X(5),
2                 A,X(4),A);
2       PUT FILE (MHSTATS) SKIP EDIT("NUM","%UTIL","NUM","%UTIL")(X(4),A,X(5),A,X(15),A,X(4),A);
2
2       end pagehead;
1
```

```
1       dcl   QSTATS   file stream output print;
1
1
1       dcl  Calc_stat  entry(fixed bin(7),fixed bin(7),float bin(31),
1                       float bin(31),float bin(31),float bin(31),
1                       float bin(31),float bin(31));
1
1       dcl   (pages,J,K,lower,upper,test) fixed bin(7);
1
1       dcl   (m1,v1,m2,v2,m3,v3 ) float bin(31);
1       dcl   reply  char(80) var;
1       open file(QSTATS);
1
1
1       if( NUMQS <= 20 ) THEN pages = 1;
1       else if( NUMQS > 20  & NUMQS <= 40 ) THEN pages = 2;
1       else if( NUMQS > 40  & NUMQS <= 60 ) then pages = 3;
1       else if( NUMQS > 60  & NUMQS <= 80 ) THEN pages = 4;
1       else pages = 5;
1
1       lower = 1;
1       upper = MIN(20,NUMQS);
1       do J = 1 to pages;
2
2               call headings;
2               do K = lower to upper;
3
3                       call Calc_stat(3,K,m1,v1,m2,v2,m3,v3);
3                       PUT FILE (QSTATS) skip edit(K,m1,v1,m2,v2)(x(10),f(3),X(6),F(5,2),
3                               X(8),F(7,3),X(7),F(3),X(6),F(3));
3                       END;
2
2                                       lower = lower + 20;
2               test = upper + 20;
2               upper = MIN(test,NUMQS);
2               end;
1
1       headings : proc;
2
2       PUT FILE (QSTATS) PAGE SKIP edit("Queue Summary Report")(X(28),A);
2       PUT FILE (QSTATS) skip edit("Queue","Average","Variance","MIN","MAX")
2                 (X(10),A,X(5),A,X(6),A,X(6),A,X(6),A);
2       PUT FILE (QSTATS) SKIP EDIT("Number","Size")(X(10),A,X(5),A);
2
2       end headings;
1
1       end QMAR;
        MHARD : proc;
1       %INCLUDE 'UCASE.PLI';
1       %INCLUDE 'TRANSPORT.PLI';
1
1       DCL  MHSTATS  FILE STREAM OUTPUT PRINT;
1
1
1       dcl  Calc_stat  entry(fixed bin(7),fixed bin(7),float bin(31),
1                       float bin(31),float bin(31),float bin(31),
```

```
1       if( v1 > 0.0 ) then v1 = 1/v1;
1       else v1 = 0.0;
1
1       put skip(2) edit('Part Production per unit time =  ',m1)(X(20),A,F(7,3));
1       put skip(2) edit('Variance                      =  ',v2)(X(20),A,F(7,3));
1       put skip(3) edit('Hit a carriage return to continue.')(X(20),A);
1       get edit(reply)(A(80));
1       call heading2;
1
1       do J = 1 to PARTS;
2
2               call Calc_stat(9,J,m1,v1,m2,v2,m3,v3);
2       if( m2 > 0.0 ) then m2 = 1/m1;
2       else m2 = 0.0;
2       if( v2 > 0.0 ) then v2 = 1/v1;
2       else v2 = 0.0;
2
2               put skip edit(PIECES.P_name(J),m2,v2,m1,v1)
2               (A(30),4(x(4),F(7,3)));
2               end;
1
1       put skip edit('Hit a carriage return to continue.')(X(20),A);
1       get edit(reply)(A(80));
1
1
1       err = '1'B;
1       do while( err );
2
2               put skip(20) edit('Would you like this report saved ?  ')(X(20),A);
2               get edit(reply)(A(80));
2               reply = UCASE(reply);
2               if( reply = 'YES' | reply = 'NO' ) then err = '0'B;
2               else put skip edit('Invalid reply')(X(20),A);
2               end;
1
1       if( reply = 'YES') then call WIPHARD;
1
1
1       heading1 : proc;
2
2       put skip(25) edit('Work in Process and Production Report')(X(17),A);
2       end heading1;
1
1       heading2 : proc;
2
2       put skip(25) edit('Production by Part Type')(X(24),A);
2       end heading2;
1
1       end WIPSTAT;

        /*****************************************************************/
        QHARD : proc;
1
1       %INCLUDE 'UCASE.PLI';
1       %INCLUDE 'Q.PLI';
1
```

```
2               put edit(reply)(A(30));
2               reply = UCASE(reply);
2               if( reply = 'YES' | reply = 'NO' ) then no = '0'B;
2               else put skip edit('Invalid reply.')(X(12),A);
2               end;
1
1       if( reply = 'YES')then call MHARD;
1
1       pagehead : proc;
2
2       put skip(25) edit('Material Handling Utilization Report')(X(18),A);
2       put skip(2) edit('PATH','MEAN','VARIANCE','PATH','MEAN','VARIANCE')(X(3),A,X(5),A,X(5),A,X(5),A,X(5),
2                   A,X(4),A);
2       PUT SKIP EDIT('NUM','%UTIL','NUM','%UTIL')(X(4),A,X(5),A,X(15),A,X(4),A);
2
2       end pagehead;
1
1
1       END MHSTAT;

        /*****************************************************************/


        /*******************************************/
        /*******************************************/

        WIPSTAT : proc;
1
1       %include 'PARTS.PLI';
1       %INCLUDE 'UCASE.PLI';
1
1
1       DCL ( m1,v1,m2,v2,m3,v3  ) float bin(31);
1       dcl  reply                 char(80) var;
1       dcl  J                     fixed bin(7);
1       DCL   err  bit(1);
1
1
1       dcl  WIPHARD   ENTRY;
1
1
1       dcl  Calc_stat   entry ( fixed bin(7),fixed bin(7),float bin(31),
1                          float bin(31),float bin(31),float bin(31),
1                          float bin(31),float bin(31));
1
1
1       call heading1;
1       put skip(3) edit('Work in Process')(X(28),A);
1       call Calc_stat(4,0,m1,v1,m2,v2,m3,v3);
1       put skip(2) edit('Average number of parts in the system =  ',m1)(X(12),A,F(6,2));
1       put skip(2) edit('Variance                              =  ',v1)(X(12),A,F(6,2));
1       CAll Calc_stat(8,0,m1,v1,m2,v2,m3,v3);
1       put skip(4) edit('System Production')(X(28),A);
1       put skip(2) edit('Average Time in the System =  ',m1)(X(20),A,F(7,3));
1       put skip(2) edit('Variance                   =  ',v1)(X(20),A,F(7,3));
1       IF( m1 > 0.0 ) then m1  = 1/m1;
1       else m1 = 0.0;
```

```
1       %INCLUDE 'TRANSPORT.PLI';
1
1       dcl  Calc_stat  entry(fixed bin(7),fixed bin(7),float bin(31),
1                            float bin(31),float bin(31),float bin(31),
1                            float bin(31),float bin(31));
1
1       dcl   (pages,J,K,lower,upper,test) fixed bin(7);
1
1       dcl   (m1,v1,m2,v2,m3,v3,m11,v11,m21,v21,m31,v31 ) float bin(31);
1
1       dcl   no  bit(1);
1       dcl    MAXCARTS   FIXED BIN(15) EXTERNAL;
1
1       dcl   reply  char(80) var;
1       dcl MHARD    ENTRY;
1
1       if( PATHS <= 20 ) THEN pages = 1;
1       else if( PATHS > 20  & PATHS <= 40 ) THEN pages = 2;
1       else if( PATHS > 40  & PATHS <= 60 ) then pages = 3;
1       else if( PATHS > 60  & PATHS <= 80 ) THEN pages = 4;
1       else pages = 5;
1
1       lower = 1;
1       upper = MIN(20,PATHS);
1       do J = 1 to pages;
2
2               call pageheaad;
2               do K = lower to upper by 2;
3
3                       call Calc_stat(5,K,m1,v1,m2,v2,m3,v3);
3                       call Calc_stat(5,K+1,m11,v11,m21,v21,m31,v31);
3                       put skip edit(K,m1,v1,m2,v2,K+1,m11,v11,m21,v21)                                    (X(4),F(3),X(5),F(5,
3                         F(5,2),X(3),F(7,3),X(5),F(3),X(5),F(5,2),
3                    X(3),F(7,3),X(5),F(5,2),X(3),F(7,3));
3
3               END;
2
2               test = upper + 20;
2               lower = lower + 20;
2               upper = MIN(test,20);
2               put skip  edit('Hit a carriage return to continue.')(X(10),A);
2               get edit(reply)(A(30));
2               end;
1
1       put skip(25) edit('CARRIER UTILIZATION')(x(25),A);
1       put skip(2) edit('Number of','Average','Variance')(X(12),A,X(12),A,X(12),A);
1       put skip edit('Carriers','Utilization')(X(12),A,X(13),A);
1       call Calc_stat(6,0,m1,v1,m2,v2,m3,v3);
1       put skip edit(MAXCARTS,m1,v1)(X(14),f(3),x(14),F(6,2),X(14),F(7,3));
1
1       put skip edit('Hit a carriage return to continue.')(X(12),A);
1
1       get edit(reply)(A(80));
1       no = '1'B;
1       do while(no);
2
2               put skip(5) edit('Would you like this report saved ?  ')(X(12),A);
```

```
  547     1                   records = 0;
  548     1
  549     1                 do  while ( more_records );
  550     2
  551     2
  552     2                  records = records + 1 ;
  553     2
  554     2                  read  file(logical) into ( RECORD )  keyto(recnum);
  555     2
  556     2                         C_name(records) =  name;
  557     2                         Nums(records)   =  caps;
  558     2                         Link(records)   =  connection;
  559     2                         Que_num(records)=  Q;
  560     2                         IO(records)     =  inout;
  561     2                         Moves(records)  =  times;
  562     2                         M_dist(records)  =  d_name ;
  563     2
  564     2                         Queues.Rank(Q) = Qrule;
  565     2                         Queues.Maxq(Q) = Qsize;
  566     2
  567     2               end;/* end of write loop  */
  568     1
  569     1                           SHUTS = records - 1;
  570     1
  571     1
  572     1                put skip(2) edit('The file has been created.') ( X(24),A);
  573     1
  574     1                     put skip edit('There were',SHUTS,'records read from the file.')(X(20),A,X(1),F(2),X(1),A);
  575     1
  576     1                     close file ( logical ) ;
  577     1
  578     1                     Call Pause;
  579     1
  580     1
  581     1
  582     1       /*********************************************************/
  583     1
  584     1       instructions : proc;
  585     2
  586     2       put skip(5) edit('++++  Filing System Instructions  ++++') ( X(21),A);
  587     2       put skip(3) edit('In the filing system you will be asked to enter') (X(15),A);
  588     2       put skip edit('a file name where your model statements can be')(X(10),A);
  589     2       put skip edit('saved.  The file name can consist of from 1 to 8')(X(10),A);
  590     2       put skip edit('letters or numbers.  However, the first character') (X(10),A);
  591     2       put skip edit('must be a letter.  Examples of valid file names')(X(10),A);
  592     2       put skip edit('are shown below.') ( X(10),A);
  593     2       put skip(2) edit('Valid File Names')(X(29),A);
  594     2       put skip(2) edit('USERFILE ')(X(33),A);
  595     2       put skip    edit('MACHINE2 ')(X(33),A);
  596     2       put skip    edit('FILE2326 ')(X(33),A);
  597     2       put skip    edit('F1234567 ')(X(33),A);
  598     2
  599     2
  600     2       END instructions;
  601     1
  602     1
```

```
490     2          get edit(reply) (A(80));
491     2          reply = UCASE(reply);
492     2
493     2          if( reply = 'Y' | reply = 'N')then error = false;
494     2
495     2           end;
496     1
497     1           if(reply = 'Y') then call instructions;
498     1
499     1           error = true;
500     1
501     1           do while( error );
502     2
503     2             put skip(5) edit('Enter a valid file name: ') (X(25),A);
504     2             get edit(filename) ( A(8));
505     2
506     2              filename = UCASE( filename );
507     2
508     2
509     2
510     2                     symbol = SUBSTR(filename,1,1);
511     2                     position = INDEX(alpha,symbol);
512     2                     if ( position = 0 ) then do;
513     3
514     3                             put skip(2) edit('The file ',filename,' is invalid, re-enter!')( A,A,A);
515     3
516     3                      error = true;
517     3                      end;
518     2
519     2
520     2                 else   error = false;
521     2
522     2                    open file ( logical )  title( filename );
523     2
524     2
525     2
526     2                        recnum = 0 ;
527     2                        read file(logical)  into (RECORD)  keyto(recnum);
528     2
529     2
530     2
531     2                     if(name ^= 'Shuttle Descriptions')then do;
532     3
533     3                             put skip edit('The file',filename,'does not contain Shuttle Descriptions.')(A,X(1),
534     3                             put skip list('You will need to enter a different filename.');
535     3                             error = true;
536     3                             close file ( logical );
537     3                             end;
538     2
539     2           end;/* end of error  do while  */
540     1
541     1
542     1
543     1                    error = false;
544     1                    put skip(5) edit('The Shuttle descriptions are being read from ',filename)(X(8),A,A(8));
545     1                    filename = filename || '.DAT' ;
```

```
  433     1
  434     1
  435     1
  436     1         dcl  alpha  char(30) var  init('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  437     1
  438     1
  439     1
  440     1
  441     1       dcl  error   bit(1),
  442     1            true    bit(1) init('1'B),
  443     1            false   bit(1) init('0'B);
  444     1
  445     1
  446     1       dcl  more_records   bit(1);
  447     1
  448     1       dcl  reply     char(80) var,
  449     1            filename  char(12)  var;
  450     1
  451     1
  452     1       dcl  recnum     fixed bin(7),
  453     1            records    fixed bin(7),
  454     1            JJ         fixed bin(7);
  455     1
  456     1       dcl   logical  file   input   keyed sequential
  457     1
  458     1       environment( maximum_record_number(100),maximum_record_size(250));
  459     1
  460     1
  461     1       on  endfile(logical) more_records = false;
  462     1       more_records = true;
  463     1
  464     1       on undefinedfile  ( logical ) begin;
  465     2
  466     2          put skip(2) edit('The file',filename,'does not exist.')(A,X(1),A(12),X(1),A);
  467     2
  468     2          error = true;
  469     2
  470     2          end;
  471     1
  472     1
  473     1         ON ANYCONDITION  BEGIN;
  474     2           PUT SKIP(3) LIST('Input/Output');
  475     2           end;
  476     1
  477     1
  478     1
  479     1       /*****************************************************/
  480     1
  481     1
  482     1       put skip(25) edit('####  FMSS Filing System  ####')(X(22),A);
  483     1
  484     1       error = true;
  485     1
  486     1       do while( error );
  487     2
  488     2         put skip(3) edit('Do you need instructions ?( Y/N ): ') (X(20),A);
```

```
  316     2       end SEARCHS;
  317     1
  318     1
  319     1       END SHUT;
  320
  321             SHUT_IN  : proc;
  322     1
  323     1       %include  'PAUSE.PLI';
  332     1
  333     1       %INCLUDE  'UCASE.PLI';
  343     1
  344     1       %INCLUDE  'FILETEST.PLI';
  365     1
  366     1
  367     1       dcl   1    Queues  ( 100 )    external,
  368     1
  369     1
  370     1                2  Head            fixed bin(15),
  371     1                2  Tail            fixed bin(15),
  372     1                2  Rank            fixed bin(15),
  373     1                2  Maxq            fixed bin(31);
  374     1
  375     1       dcl    NUMQS        fixed bin(7)  external;
  376     1
  377     1
  378     1
  379     1       %include 'SHUTS.PLI';
  404     1
  405     1
  406     1
  407     1
  408     1
  409     1       dcl  1   RECORD    ,
  410     1
  411     1
  412     1                          2   name    char (30),
  413     1                          2   caps    fixed bin(7),
  414     1                          2   connection  char(30),
  415     1                          2   Q           fixed bin(7),
  416     1                          2   inout         char(1),
  417     1                          2   times       fixed bin(7),
  418     1                          2   Qrule        fixed bin(7),
  419     1                          2   Qsize        fixed bin(15),
  420     1                          2   d_name       char(15) var;
  421     1
  422     1
  423     1       dcl  new_rule    char(80) var;
  424     1
  425     1       dcl  convert     entry(fixed bin(7),char(80) var);
  426     1
  427     1       dcl  Que  entry  ( char(80) var,fixed bin(7));
  428     1
  429     1
  430     1       dcl  symbol char(3) var;
  431     1
```

```
  259     2       put skip edit('the second section of input is a " > " .')(X(20),A);
  260     2
  261     2
  262     2       end instructions;
  263     1
  264     1       /*****************************************************************/
  265     1
  266     1       SEARCHS : PROC;
  267     2
  268     2       dcl   duplicate     bit(1),
  269     2             tru           bit(1) init('1'B),
  270     2             no            bit(1) init('0'B),
  271     2             err1          bit(1);
  272     2
  273     2       dcl   j             fixed bin(7);
  274     2             k             fixed bin(7);
  275     2
  276     2
  277     2       dcl answer         char(15) var;
  278     2
  279     2
  280     2       duplicate = tru;
  281     2
  282     2       do while ( duplicate );
  283     3
  284     3          err1 = no;
  285     3          j    = 0;
  286     3
  287     3          do while( j < (SHUTS - 1 ) & ( ^err1));
  288     4
  289     4             j = j + 1 ;
  290     4             if( C_name(j) = string1 ) then err1 = tru;
  291     4
  292     4             end;
  293     3
  294     3             if( err1 ) then call new_name;
  295     3
  296     3             else
  297     3                  duplicate = no;
  298     3
  299     3             end;
  300     2
  301     2
  302     2          /*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*/
  303     2
  304     2
  305     2        new_name : proc;
  306     3
  307     3            put skip(2) edit('@@@@  Duplicate Shuttle Name  @@@@')(X(5),A);
  308     3            put skip(2) edit('Shuttle',j,'is the same as Shuttle',SHUTS)(X(5),A,X(1),F(2),X(1),A,X(1),F(2));
  309     3
  310     3            put skip(2) edit('Enter a new name for Shuttle',SHUTS,'    ')(A,F(2),A);
  311     3
  312     3                        get edit(string1)(A(80));
  313     3                        string1 = UCASE(string1);
  314     3                         end new_name;
```

```
202     3               string6 = UCASE( string6 );
203     3
204     3
205     3               Call  Parm_in(string6,pointer);
206     3               Movas(SHUTS) = pointer;
207     3
208     3               M_dist( SHUTS ) = string6 ;
209     3
210     3               put skip(2) list("Shuttle name:  ");
211     3               get list( string1 );
212     3               string1 = UCASE(string1);
213     3
214     3             End;  /* end of QUIT Loop  */
215     2
216     2
217     2               filerr  =  true;
218     2
219     2               do while ( filerr );
220     3
221     3                   put skip(3) list("Would you like your descriptions saved in a file:( Y/N )?  ");
222     3                   get list(string7);
223     3                       string7 = UCASE(string7);
224     3                   if( string7 = "Y" | string7  = "N")then filerr = false;
225     3                   end;
226     2
227     2               if( string7 = "Y" ) then call SAVE_SHT;
228     2
229     2             end; /* end of new loop  */
230     1
231     1
232     1        /*********************************************************/
233     1
234     1       do while( old );
235     2
236     2             old = false;
237     2
238     2             Call SHUT_IN;
239     2
240     2             end;
241     1
242     1
243 |   1       /*************************************************************/
244     1
245     1
246     1       instructions : proc;
247     2
248     2       put skip(3) edit("####  Shuttle Input Instructions  ####")(X(20),A);
249     2       put skip(3) edit("The shuttle input description is divided into two")(X(20),A);
250     2       put skip edit("sections.  The first section asks you for a shuttle")(X(15),A);
251     2       put skip edit("name.  You may either input a name less than 30")(X(15),A);
252     2       put skip edit("characters or type QUIT to terminate the shuttle ")(X(15),A);
253     2       put skip edit("input sequence. ")(X(15),A);
254     2       put skip edit("The second section will ask you to input six ")(X(20),A);
255     2       put skip edit("different items.  The six input items are as ")(X(15),A);
256     2       put skip edit("follows:  Shuttle Capacity, Queue Rank, Queue Disipline, ")(X(15),A);
257     2       put skip edit("Work Center link, Shuttle funciton: input/output, and a ")(X(15),A);
```

```
  145     3              get list(string2,string3,string7,string4,string5,string6);
  146     3
  147     3              numerr = false;
  148     3              Call lex2(string2,caps,numerr);
  149     3
  150     3              do while ( numerr );
  151     4
  152     4                 put skip(2) list('The capacity must be numeric, re-enter: ');
  153     4                 get list(string2);
  154     4
  155     4                 numerr = false;
  156     4                 Call lex2(string2,caps,numerr);
  157     4                 end;
  158     3
  159     3            Nums(SHUTS) = CEIL(caps);
  160     3
  161     3          numerr = false;
  162     3
  163     3            call lex2(string7,caps,numerr);
  164     3
  165     3                    if( caps > 100 | caps < 1 ) then numerr = '1'B;
  166     3
  167     3          do while(numerr);
  168     4
  169     4             put skip(2) list('The Q number is invalid, re-enter:  ');
  170     4             get edit(string7)(A(80));
  171     4             numerr = false;
  172     4             call lex2( string7,caps,numerr);
  173     4
  174     4                if(  caps > 100 | caps < 1 ) then numerr = '1'B;
  175     4
  176     4             end;
  177     3
  178     3
  179     3
  180     3
  181     3                pointer = CEIL( caps );
  182     3
  183     3            string3 = UCASE(string3 );
  184     3            Call QUE(string3,pointer );
  185     3
  186     3            Que_num(SHUTS) = pointer;
  187     3            Link(SHUTS) = UCASE(string4);
  188     3
  189     3            string5 = UCASE(string5);
  190     3            IOerr = true;
  191     3            if( string5 = 'I'  | string5  = 'O' ) then IOerr = false;
  192     3            do while ( IOerr );
  193     4
  194     4               put skip(2) list('The shuttle was not specified as Input or Output !');
  195     4               put skip list('Re-enter: ');
  196     4               get list(string5);
  197     4                  string5 = UCASE(string5);
  198     4               if( string5  = 'I' | string5  = 'O' )then IOerr = false;
  199     4               end;
  200     3
```

```
     1
     1
     1        /*****************************
     1        */
     1
     1        put skip(25);
     1        put skip edit('!!!!  SHUTTLE  DESCRIPTIONS  !!!!') ( X(20),A);
     1
     1        old = false;   new = false;
     1
     1        err = true;
     1
     1        do while ( err );
     2
     2           put skip(3) list('Do you wish to use NEW or EXISTING Shuttle descriptions?');
     2           put skip list('Reply ( NEW/OLD ): ');
     2           GET list(string1);
     2           string1 = UCASE(string1);
     2
     2           if(string1 = 'NEW' | string1 = 'OLD' ) then err = false;
     2
     2           end;
     1
     1        if ( string1 = 'NEW' ) then new = true;
     1
     1         else
     1                old = true;
     1
     1
     1        do while(new );
     2
     2          put skip(5) list('Do you need to see the instruction set? ( YES/NO ): ');
     2          get list(string1);
     2          string1 = UCASE(string1);
     2
     2          if(string1 = 'YES' | string1 = 'Y' ) then call instructions;
     2
     2           new = false;
     2           string1 = '';
     2           SHUTS = 0;
     2
     2            put skip(2) list('Shuttle name: ');
     2            get edit(string1) ( A(30));
     2            string1 = UCASE(string1);
     2
     2            do while (string1 ^=  'QUIT');
     3                      SHUTS = SHUTS + 1 ;
     3
     3
     3
     3               C_name(SHUTS) = string1;
     3
     3               put skip(2) list('Capacity , Q Rank , Q number , Work Center , I/O , Distribution ');
     3
```

```
                SHUT  : proc;

                %include "SHUTs.PLI";

                /*************************/

                dcl lex2  entry   ( char(80) var, float bin(31), bit(1));

                dcl  Parm_in  entry      ( char(80) var, fixed bin(7));

                dcl  Que   entry  ( char(80) var, fixed bin(7));

                dcl  SAVE_SHT         entry;

                dcl  SHUT_IN          entry;

                %include  "UCASE.PLI";

                /*************************/



                /*******************************/

                dcl  (   string1,
                         string2,
                         string3,
                         string4,
                         string5,
                         string6,
                         string7 )          char (80) var;

                dcl  value    float bin(31),
                     caps     float bin(31);

                dcl  pointer fixed bin(7);

                dcl (  true  bit(1)  init('1'b),
                       false bit(1)  init('0'B),

                       err   bit(1),
                       new   bit(1),
                       old   bit(1),

                       numerr bit(1),
                       IJerr  bit(1),
```

```
1304  5          PATHS = PATHS + 1 ;
1305  5          put skip(2) edit('Enter the end point, the path length, and the travel rate for path ',K)(A,F
1306  5          put skip(2) edit('*     ')(A);
1307  5          get list(Endpoint,distance,dist);
1308  5          Endpoint = UCASE(Endpoint);
1309  5          err = true;
1310  5
1311  5
1312  5                do while ( err );
1313  6
1314  6             do L = 1 to works by 1 ;
1315  7
1316  7                if( M_name(L) = Endpoint)then err = false;
1317  7                end;
1318  6
1319  6             if ( err )then do;
1320  7
1321  7                put skip(2) edit(Endpoint,'is not a valid work center name.')(X(2),A,A);
1322  7
1323  7                   /*********  fix on 12/30/83 ******/
1324  7
1325  7                   put skip(2) edit('The following are valid work center names:')(X(10),A);
1326  7
1327  7                   do L = 1 to works by 2;
1328  8
1329  8                   put skip edit( M_name(L),M_name(L+1))
1330  8                                 (X(10),A(30),X(10),A(30));
1331  8                    end;
1332  7
1333  7                put skip edit('Enter a new name :  ')(X(5),A);
1334  7                get list(Endpoint);
1335  7                  Endpoint = UCASE(Endpoint);
1336  7                end;
1337  6
1338  6          end;
1339  5
1340  5                err = true;
1341  5
1342  5                do while ( err );
1343  6                   err = false;
1344  6
1345  6                   call lex2(distance,value,err );
1346  6                   if ( err ) then do;
1347  7
1348  7                       put skip(2) edit('The path length must be numeric !')(X(2),A);
1349  7                      put skip edit('Re-enter :  ')(X(2),A);
1350  7                      get edit (distance)(A(80));
1351  7                      end;
1352  6
1353  6                 end;
1354  5
1355  5
1356  5                     dist = UCASE(dist);
1357  5
1358  5                call Parm_in(dist , pointer );
1359  5
```

```
1247    1
1248    1
1249    1       else  WCD = true;
1250    1       do while( WCD );
1251    2
1252    2          WCD = false;
1253    2
1254    2
1255    2       PUT SKIP(25) EDIT('<<<<  Material Handling System Description  >>>>')(X(12),A);
1256    2       put skip(3) edit('Would you like to use NEW or OLD Material Handling Sytem Descriptions?')(A);
1257    2       put skip edit('Reply (NEW/OLD) :  ')(A);
1258    2
1259    2       err = true;
1260    2       do while (err);
1261    3
1262    3          get edit(string1)(A(80));
1263    3          string1 = UCASE(string1);
1264    3          if (string1 ^= 'NEW' & string1 ^= 'OLD')then put skip list('Invalid, re-enter :  ');
1265    3          else
1266    3              err = false;
1267    3          end;
1268    2
1269    2       if string1 = 'NEW' then  new = true;
1270    2
1271    2        else old = true;
1272    2
1273    2        do while ( new );
1274    3
1275    3          new  = false;
1276    3
1277    3
1278    3
1279    3          put skip(25) edit('<<<<  Material Handling System Description  >>>>')(X(12),A);
1280    3
1281    3          PATHS = 0;
1282    3       DO  J = 1 to works by 1;
1283    4
1284    4       Current =  Machine.M_name(J);
1285    4
1286    4
1287    4              PUT SKIP(2) edit('How many paths emanate from station ',Current,'  ')(A,A,A,A);
1288    4
1289    4              err = true;
1290    4              do while (err );
1291    5
1292    5                   get edit(string1)(A(80));
1293    5                  call lex2(string1,value,err);
1294    5
1295    5                 if( err )then put skip(2) edit('Invalid input, re-enter:')(x(2),A);
1296    5
1297    5
1298    5                 end;
1299    4
1300    4                     branches = CEIL( value );
1301    4
1302    4              do  K = 1 to branches by 1;
```

```
1173    1               Pend            char(30) var,
1174    1               Current         char(30) var,
1175    1               Endpoint        char(30) var,
1176    1               distance        char(80) var,
1177    1               string1         char(80) var,
1178    1               name            char(30) var,
1179    1               dist            char(80) var;
1180    1
1181    1       dcl     J               fixed bin(7),
1182    1               K               fixed bin(7),
1183    1               L               fixed bin(7),
1184    1               pointer         fixed bin(7),
1185    1               value           float bin(31),
1186    1               branches        fixed bin(7),
1187    1               elements        fixed bin(7);
1188    1
1189    1       dcl     WCD             bit(1) init('0'B),
1190    1               err             bit(1) init('0'B),
1191    1               true    bit(1) init('1'B),
1192    1               false   bit(1) init('0'B),
1193    1               STCKCNT          bit(1) init('0'B);
1194    1
1195    1       dcl       new     bit(1) init('0'B),
1196    1                 old     bit(1) init('0'B);
1197    1
1198    1
1199    1
1200    1       dcl     DONE(25)        char(30) var,
1201    1               nums            fixed bin(7),
1202    1               before          bit(1);
1203    1
1204    1
1205    1       dcl     lex2              entry     (char(80) var,float bin(31),bit(1));
1206    1
1207    1       dcl         MHS_IN        entry;
1208    1
1209    1       dcl         MHS_SAVE      entry;
1210    1
1211    1
1212    1       %include 'PAUSE.PLI';
1221    1
1222    1       dcl     Parm_in         entry       (char(80) var, fixed bin(7));
1223    1
1224    1       %include  'UCASE.PLI';
1234    1
1235    1
1236    1       /*****************************************************************************************/
1237    1
1238    1
1239    1       IF  (works <= 0 ) then  do;
1240    2
1241    2               WCD = false;
1242    2               put skip(2) list('You need to enter the Work Center Descriptions before');
1243    2               put skip list('you enter the Material Handling System Description.');
1244    2                Call Pause;
1245    2
```

```
 1072    4
 1073    4            put skip(5) edit("Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  ")(A);
 1074    4          get edit (ans) (A(80));
 1075    4          err = false;
 1076    4
 1077    4          call lex2( ans,val,err );
 1078    4          if( err ) then put skip list("Invalid entry");
 1079    4          end;
 1080    3
 1081    3          if ( val = 0.0 ) then c = 32767;
 1082    3
 1083    3          else   c = CEIL(val);
 1084    3
 1085    3
 1086    3                END; /* END OF ELSE DO   */
 1087    2
 1088    2             Head(B) = 0;
 1089    2             Tail(B) = 0;
 1090    2             Maxq(B) = c;
 1091    2
 1092    2       end initq;
 1093    1
 1094    1
 1095    1     End  Que;
 1096
 1097          convert : proc ( rank, rule);
 1098    1
 1099    1     dcl  rank  fixed bin(7),
 1100    1          rule  char(80) var;
 1101    1
 1102    1
 1103    1           if( rank = 1 ) then rule = "FIFO";
 1104    1
 1105    1     else  if( rank = 2 ) then rule = "LIFO";
 1106    1
 1107    1     else  if( rank = 3 ) then rule = "SPT";
 1108    1
 1109    1     else  if( rank = 4 ) then rule = "LNRO";
 1110    1
 1111    1     else  rule = "RAND";
 1112    1
 1113    1     end convert;
 1114          TRANS :  PROC;
 1115    1
 1116    1
 1117    1     /*****************************************************************************/
 1118    1
 1119    1
 1120    1     %INCLUDE  "MACHINE.PLI";
 1144    1
 1145    1
 1146    1
 1147    1
 1148    1     %INCLUDE "TRANSPORT.PLI";
 1170    1
 1171    1
```

```
1015  2       dcl val  float bin(31);
1016  2
1017  2       IF (Queues.Maxq(3) = 0 ) then do;
1018  3
1019  3                 if(NUMQS = 100) then do;
1020  4                           Call ERROR(201,' ');
1021  4                           NUMQS = 100;
1022  4                           end;
1023  3
1024  3                 else NUMQS = NUMQS + 1;
1025  3
1026  3
1027  3       put skip list('debug for queue counter ',NUMQS);
1028  3
1029  3                 END;
1030  2
1031  2
1032  2
1033  2
1034  2        if( Maxq(8) > 0 ) then do;
1035  3
1036  3           put skip(2) edit('Queue number ',8,' already has a capacity of ',Maxq(8))(A,F(3),A,F(5));
1037  3           put skip edit('Would you like to change that capacity? (Y/N): ')(A);
1038  3           get edit(ans)(A(80));
1039  3           ans = UCASE(ans);
1040  3           if( ans = 'Y')then do;
1041  4
1042  4
1043  4         err = true;
1044  4
1045  4         do while ( err );
1046  5
1047  5             put skip(5) edit('Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  ')(A);
1048  5           get edit (ans) (A(80));
1049  5           err = false;
1050  5
1051  5           call lex2( ans,val,err );
1052  5           if( err ) then put skip list('Invalid entry');
1053  5           end;
1054  4
1055  4           if ( val = 0.0 ) then c = 32767;
1056  4
1057  4           else    c = CEIL(val);
1058  4
1059  4               end; /* end of second if do */
1060  3
1061  3
1062  3               ELSE    c = Maxq(8);
1063  3
1064  3             END;/* END OF FIRST IF DO */
1065  2
1066  2       ELSE DO;
1067  3
1068  3
1069  3         err = true;
1070  3
```

```
   958     2            if(temp = 1 ) then old = 'FIFO';
   959     2
   960     2            else if(temp = 2)then old = 'LIFO';
   961     2
   962     2            else if(temp = 3)then old = 'SPT';
   963     2
   964     2            else if(temp = 4)then old = 'LNRO';
   965     2
   966     2            else  old = 'RAND';
   967     2
   968     2            IF( rank ^= old  ) then do;
   969     3
   970     3                put skip(2) edit('Queue number ',number,' has a ranking criteria of ',old)(A,F(2),A,A);
   971     3                put skip edit('You specified the new criteria as ',rank)(A,A);
   972     3                put skip edit('Would you like the "NEW" or "OLD" ranking criteria ?')(A);
   973     3                error = true;
   974     3                do while ( error );
   975     4
   976     4                   put skip edit('Enter :(NEW/OLD) ?  ')(A);
   977     4                   get edit( ans ) (A(80));
   978     4                   ans = UCASE(ans);
   979     4                   if(ans = 'NEW' | ans = 'OLD')then error = false;
   980     4
   981     4                    else put skip list('Invalid');
   982     4                    end;
   983     3
   984     3                if( ans = 'NEW')then Queues.Rank(number) = rank1;
   985     3                else
   986     3                     Queues.Rank(number) = temp;
   987     3
   988     3              end; /* end of second if do */
   989     2
   990     2          Else      Queues.Rank(number) = temp;
   991     2
   992     2          end;/*  end of first if do  */
   993     1
   994     1        ELSE    Queues.Rank(number) = rank1;
   995     1
   996     1        call initq(number);
   997     1
   998     1        PUT SKIP LIST('SHUT DEBUG FOR NUMQS  QNUM = ',number,NUMQS);
   999     1
  1000     1
  1001 |   1        /*********************************************************/
  1002     1
  1003     1
  1004     1        initq : proc (  B );
  1005     2
  1006     2        dcl ERROR  entry(fixed bin(15) , char (80) var);
  1007     2
  1008     2        dcl lex2  entry(char(80) var, float bin(31), bit(1));
  1009     2
  1010     2        dcl     A    fixed bin(7);
  1011     2        dcl     B    fixed bin(7);
  1012     2        dcl     c    fixed bin(31);
  1013     2
```

```
901     3                                   error = false;
902     3                                   rank1 = 1;
903     3                                    end;
904     3
905     2
906     2          else
907     2               if( rank = 'LIFO') then do;
908     3
909     3                                   error = false;
910     3                                   rank1 = 2;
911     3                                   end;
912     2
913     2          else
914     2               if( rank = 'SPT')  then  do;
915     3
916     3                                    error = false;
917     3                                   rank1 = 3;
918     3                                   end;
919     2
920     2          else
921     2               if( rank = 'LNRO') then  do;
922     3
923     3                                    error = false;
924     3                                   rank1 = 3;
925     3                                   end;
926     2
927     2          else
928     2               if( rank = 'RAND') then do;
929     3
930     3                                   error = false;
931     3                                   rank1 = 4;
932     3                                   end;
933     2
934     2          else
935     2               Do;
936     3
937     3                   put skip(2) edit('the queueing discipline entered is invalid.')(X(3),A);
938     3                   put skip(3) edit('The valid queueing rulas are as follows:')(X(3),A);
939     3                   put skip(2) edit('FIFO   ==>  First in- First out')(X(8),A);
940     3                   put skip edit('LIFO   ==>  Last in- First out')(X(8),A);
941     3                   put skip edit('SPT    ==>  Shortest processing time first')(X(8),A);
942     3                   put skip edit('LNRO   ==>  Least Number of Remaining Operations')(X(8),A);
943     3                   put skip edit('RAND   ==>  Random order')(X(8),A);
944     3
945     3                   put skip edit('Select a valid queueing rule:  ')(A);
946     3                   get list (rank);
947     3                   rank = UCASE(rank);
948     3                   error = true;
949     3                   END; /*                ENC OF INVALID INPUT LOOP  */
950     2
951     2
952     2               END;  /* END OF MAIN LOOP */
953     1
954     1
955     1          if( Queues.Rank(number) > 0 ) then do;
956     2
```

```
835     2
836     2       put skip(5) edit("++++  Filing System Instructions  ++++") ( X(21),A);
837     2       put skip(3) edit("In the filing system you will be asked to enter") (X(15),A);
838     2       put skip edit("a file name where your model statements can be")(X(10),A);
839     2       put skip edit("saved.  The file name can consist of from 1 to 8")(X(10),A);
840     2       put skip edit("letters or numbers.  However, the first character") (X(10),A);
841     2       put skip edit("must be a letter.  Examples of valid file names")(X(10),A);
842     2       put skip edit("are shown below.") ( X(10),A);
843     2       put skip(2) edit("Valid File Names")(X(29),A);
844     2       put skip(2) edit("USERFILE ")(X(33),A);
845     2       put skip    edit("MACHINE2 ")(X(33),A);
846     2       put skip    edit("FILE2326 ")(X(33),A);
847     2       put skip    edit("F1234567" )(X(33),A);
848     2
849     2
850     2       END instructions;
851     1
852     1
853     1       END  SAVE_SHT;
854             Que : proc ( rank, number );
855     1
856     1       %INCLUDE  'UCASE.PLI';
866     1
867     1
868     1       dcl   1    Queues  ( 100 )    external,
869     1
870     1
871     1                2  Head            fixed bin(15),
872     1                2  Tail            fixed bin(15),
873     1                2  Rank            fixed bin(15),
874     1                2  Maxq            fixed bin(31);
875     1
876     1       dcl    NUMQS        fixed bin(7)  external;
877     1
878     1
879     1       dcl rank  char(80) var;
880     1       dcl  ans       char(80) var,
881     1            temp      fixed bin(15);
882     1
883     1       dcl  old       char(80) var;
884     1
885     1       dcl number fixed bin(7);
886     1
887     1       dcl error      bit(1),
888     1           true       bit(1)  init('1'B),
889     1           false      bit(1)  init('0'B),
890     1           err        bit(1);
891     1
892     1       dcl rank1      fixed bin(7);
893     1
894   | 1       /*********************************/
895     1
896     1       error = true  ;
897     1
898     1       do while ( error );
899     2
```

```
778     1
779     1
780     1
781     1                  error = false;
782     1                  put skip(10) edit('The Shuttle descriptions are being written into ',filename)(X(9),A,A(8));
783     1                  filename = filename || '.DAT' ;
784     1
785     1
786     1
787     1
788     1
789     1                        open file( OUTFILE )  output  title(filename);
790     1
791     1                    recnum =  1;
792     1
793     1
794     1                    name = 'Shuttle Descriptions';
795     1                    caps = SHUTS;
796     1                    write file(OUTFILE)  from(RECORD)   keyfrom(recnum);
797     1
798     1
799     1                  do JJ = 1 to SHUTS;
800     2
801     2                   recnum = recnum + 1 ;
802     2
803     2                         name =  C_name(JJ);
804     2                         caps =  Nums (JJ);
805     2                         connection = Link(JJ);
806     2                         Q          = Que_num(JJ);
807     2                         inout      = IO(JJ);
808     2                         times      =  Moves(JJ);
809     2                         Qrule      =  Queues.Rank(Que_num(JJ));
810     2                           Qsize     =  Queues.Maxq(Que_num(JJ));
811     2                         d_name     = M_dist( JJ );
812     2
813     2
814     2
815     2
816     2       write file ( OUTFILE ) from (RECORD)  keyfrom(recnum );
817     2
818     2
819     2              end;/* end of write loop  */
820     1
821     1               put skip(2) edit('The file has been created.') ( X(24),A);
822     1
823     1                       put skip edit('There were',SHUTS,'records written.')(X(23),A,X(1),F(2),X(1),A);
824     1
825     1
826     1                CLOSE FILE( OUTFILE );
827     1
828     1                         call Pause;
829     1
830     1
831     1
832 |   1       /*******************************************************/
833     1
```

```
721     1            filename  char(12)  var;
722     1
723     1
724     1       dcl  recnum      fixed bin(7);
725     1            JJ          fixed bin(7);
726     1
727     1       dcl  OUTFILE  file        keyed environment(
728     1
729     1                                   maximum_record_number(100),
730     1                                         maximum_record_size(250));
731     1
732     1
733     1
734     1
735     1       /*****************************************************/
736     1
737     1
738     1       put skip(25) edit("####  FMSS Filing System  ####")(X(22),A);
739     1
740     1       error = true;
741     1
742     1       do while( error );
743     2
744     2         put skip(3) edit("Do you need instructions ?( Y/N ): ") (X(20),A);
745     2
746     2         get edit(reply) (A(80));
747     2         reply = UCASE(reply);
748     2
749     2         if( reply = "Y" | reply = "N")then error = false;
750     2
751     2          end;
752     1
753     1          if(reply = "Y") then call instructions;
754     1
755     1          error = true;
756     1
757     1          do while( error );
758     2
759     2            put skip(5) edit("Enter a valid file name: ") (X(25),A);
760     2            get edit(filename) ( A(8));
761     2
762     2             filename = UCASE( filename );
763     2
764     2
765     2
766     2                    symbol = SUBSTR(filename,1,1);
767     2                    position = INDEX(alpha,symbol);
768     2                    if ( position = 0 ) then do;
769     3
770     3                     put skip(2) edit("The file",filename,"is invalid, re-enter! ")(x(18),A,x(1),A,A);
771     3                     error = true;
772     3                     end;
773     2
774     2
775     2              else   error = false;
776     2
```

```
SAVE_SHT  :  proc;

%include "PAUSE.PLI";

%INCLUDE  "UCASE.PLI";

%INCLUDE  "FILETEST.PLI";


%INCLUDE  "SHUTS.PLI";


DCL    1    RECORD,

            2  name    char(30),
            2  caps    fixed bin(7),
            2  connection    char(30),
            2  Q             fixed bin(7),
            2  inout         char(1),
            2  times         fixed bin(7),
            2  Qrule         fixed bin(7),
            2  Qsize         fixed bin(15),
            2  d_name        char(15) var;

dcl   1    Queues  ( 100 )    external,

          2  Head           fixed bin(15),
          2  Tail           fixed bin(15),
          2  Rank           fixed bin(15),
          2  Maxq           fixed bin(31);

dcl    NUMJS       fixed bin(7)  external;

dcl  symbol char(3) var;

dcl position fixed bin(7);


  dcl  alpha  char(30) var  init("ABCDEFGHIJKLMNOPQRSTUVWXYZ");


dcl  error   bit(1),
     true    bit(1) init('1'B),
     false   bit(1) init('0'B);
```

```
1819  3                     end;
1820  2
1821  2
1822  2               else   error = false;
1823  2
1824  2          end;/* end of error  do while  */
1825  1
1826  1
1827  1
1828  1               error = false;
1829  1               put skip(10) edit('The Material Handling System descriptions are being written into ',filenam
1830  1               filename = filename || '.DAT' ;
1831  1
1832  1
1833  1
1834  1
1835  1
1836  1                     open file( OUTFILE )  output  title(filename);
1837  1
1838  1                 recnum =  1;
1839  1
1840  1
1841  1                 Begin = 'MHS Descriptions';
1842  1                 time = PATHS;
1843  1                 write file(OUTFILE)  from(RECORD)   keyfrom(recnum);
1844  1
1845  1
1846  1               do JJ = 1 to PATHS;
1847  2                     recnum = recnum + 1;
1848  2
1849  2                     Begin = Zones(JJ).Start;
1850  2                     distance  = Zones(JJ).Length;
1851  2                     time      = Zones(JJ).Rate;
1852  2                     dist      = Zones(JJ).D_name;
1853  2                     Finish    = Zones(JJ).End;
1854  2
1855  2
1856  2
1857  2
1858  2
1859  2
1860  2     write file ( OUTFILE ) from (RECORD)  keyfrom(recnum );
1861  2
1862  2
1863  2              end;/* end of write loop  */
1864  1
1865  1               put skip(2) edit('The file has been created.') ( X(24),A);
1866  1
1867  1                     put skip edit('There were',PATHS,'records written.')(X(23),A,X(1),F(2),X(1),A);
1868  1
1869  1
1870  1                CLOSE FILE( OUTFILE );
1871  1
1872  1                     call Pause;
1873  1
1874  1
```

```
1762    1
1763    1       dcl  error   bit(1),
1764    1            true    bit(1) init('1'B),
1765    1            false   bit(1) init('0'B);
1766    1
1767    1       dcl  reply     char(80) var,
1768    1            filename  char(12)  var;
1769    1
1770    1
1771    1       dcl  recnum     fixed bin(7),
1772    1            JJ         fixed bin(7);
1773    1
1774    1       dcl  OUTFILE  file         keyed environment(
1775    1
1776    1                                    maximum_record_number(100),
1777    1                                          maximum_record_size(250));
1778    1
1779    1
1780    1
1781    1
1782 |  1       /*****************************************************/
1783    1
1784    1
1785    1       put skip(25) edit("####  FMSS Filing System  ####")(X(22),A);
1786    1
1787    1       error = true;
1788    1
1789    1       do while( error );
1790    2
1791    2         put skip(3) edit("Do you need instructions ?( Y/N ): ") (X(20),A);
1792    2
1793    2         get edit(reply) (A(80));
1794    2         reply = UCASE(reply);
1795    2
1796    2         if( reply = "Y" | reply = "N")then error = false;
1797    2
1798    2          end;
1799    1
1800    1          if(reply = "Y") then call instructions;
1801    1
1802    1          error = true;
1803    1
1804    1          do while( error );
1805    2
1806    2            put skip(5) edit("Enter a valid file name: ") (X(25),A);
1807    2            get edit(filename) ( A(8));
1808    2
1809    2             filename = UCASE( filename );
1810    2
1811    2
1812    2
1813    2                    symbol = SUBSTR(filename,1,1);
1814    2                    position = INDEX(alpha,symbol);
1815    2                    if ( position = 0 ) then do;
1816    3
1817    3                     put skip(2) edit("The file",filename,"is invalid, re-enter! ")(x(18),A,x(1),A,A);
```

```
1647    2
1648    2       put skip(5) edit('++++  Filing System Instructions  ++++') ( X(21),A);
1649    2       put skip(3) edit('In the filing system you will be asked to enter') (X(15),A);
1650    2       put skip edit('a file name where your model statements can be')(X(10),A);
1651    2       put skip edit('retrieved.  The file name can consist of from 1 to 8')(X(10),A);
1652    2       put skip edit('letters or numbers.  However, the first character') (X(10),A);
1653    2       put skip edit('must be a letter.  Examples of valid file names')(X(10),A);
1654    2       put skip edit('are shown below.') ( X(10),A);
1655    2       put skip(2) edit('Valid File Names')(X(29),A);
1656    2       put skip(2) edit('USERFILE ')(X(33),A);
1657    2       put skip    edit('MACHINE2 ')(X(33),A);
1658    2       put skip    edit('FILE2326 ')(X(33),A);
1659    2       put skip    edit('F1234567' )(X(33),A);
1660    2
1661    2
1662    2       END instructions;
1663    1
1664    1
1665    1       end  MMS_IN;
1666             MMS_SAVE  :  proc;
1667    1
1668    1
1669    1       %include 'PAUSE.PLI';
1678    1
1679    1       %INCLUDE  'UCASE.PLI';
1689    1
1690    1       %INCLUDE  'FILETEST.PLI';
1711    1
1712    1
1713    1
1714    1
1715    1       %INCLUDE  'TRANSPORT.PLI';
1737    1
1738    1
1739    1
1740    1
1741    1
1742    1
1743    1       dcl  1   RECORD    ,
1744    1
1745    1
1746    1                2   Begin       char(30) var,
1747    1                2   distance    float bin(31),
1748    1                2   time        fixed bin(31),
1749    1                2   dist        char(15) var,
1750    1                2   Finish      char(30) var;
1751    1
1752    1       dcl  symbol char(3) var;
1753    1
1754    1       dcl position fixed bin(7);
1755    1
1756    1
1757    1
1758    1         dcl  alpha  char(30) var  init('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
1759    1
1760    1
```

```
1590    2                       recnum = 0 ;
1591    2                       read file(logical)  into (RECORD)  keyto(recnum);
1592    2
1593    2
1594    2
1595    2                   if(Begin ^= 'MHS Descriptions')then do;
1596    3
1597    3                           put skip edit('The file',filename,'does not contain Material Handling System Descri
1598    3                           put skip list('You will need to enter a different filename.');
1599    3                           error = true;
1600    3                           close file ( logical );
1601    3                           end;
1602    2
1603    2           end;/* end of error  do while  */
1604    1
1605    1
1606    1
1607    1                   error = false;
1608    1                   put skip(5) edit('The Material Handling Sytem descriptions are being read from ',filename)(X(
1609    1                   filename = filename || '.DAT' ;
1610    1
1611    1                     records = 0;
1612    1
1613    1                   do  while ( more_records );
1614    2
1615    2
1616    2                    records = records + 1 ;
1617    2
1618    2                    read  file(logical) into ( RECORD )  keyto(recnum);
1619    2
1620    2                  Zones(records).Start = Begin;
1621    2                  Zones(records).Length = distance;
1622    2                  Zones(records).Rate   = time;
1623    2                  Zones(records).D_name = dist;
1624    2                  Zones(records).End    = Finish;
1625    2
1626    2
1627    2
1628    2               end;/* end of write loop  */
1629    2
1630    1
1631    1                           PATHS = records - 1;
1632    1
1633    1
1634    1                put skip(2) edit('The file has been created.') ( X(24),A);
1635    1
1636    1                     put skip edit('There were',PATHS,'records read from the file.')(X(20),A,X(1),F(2),X(1),A);
1637    1
1638    1                     close file ( logical ) ;
1639    1
1640    1                     Call Pause;
1641    1
1642    1
1643    1
1644    1   /*********************************************************/
1645    1
```

```
1533    2
1534    2          end;
1535    1
1536    1
1537    1        ON ANYCONDITION  BEGIN;
1538    2          PUT SKIP(3) LIST('Input/Output Error');
1539    2          end;
1540    1
1541    1
1542    1
1543 |  1      /*****************************************************/
1544    1
1545    1
1546    1      put skip(25) edit('####  FMSS Filing System  ####')(X(22),A);
1547    1
1548    1      error = true;
1549    1
1550    1      do while( error );
1551    2
1552    2        put skip(3) edit('Do you need instructions ?( Y/N ): ') (X(20),A);
1553    2
1554    2        get edit(reply) (A(80));
1555    2        reply = UCASE(reply);
1556    2
1557    2        if( reply = 'Y' | reply = 'N')then error = false;
1558    2
1559    2         end;
1560    1
1561    1         if(reply = 'Y') then call instructions;
1562    1
1563    1         error = true;
1564    1
1565    1         do while( error );
1566    2
1567    2           put skip(5) edit('Enter a valid file name: ') (X(25),A);
1568    2           get edit(filename) ( A(8));
1569    2
1570    2            filename = UCASE( filename );
1571    2
1572    2
1573    2
1574    2                  symbol = SUBSTR(filename,1,1);
1575    2                  position = INDEX(alpha,symbol);
1576    2                  if ( position = 0 ) then do;
1577    3
1578    3                           put skip(2) edit('The file ',filename,' is invalid, re-enter!')( A,A,A);
1579    3
1580    3                   error = true;
1581    3                   end;
1582    2
1583    2
1584    2               else   error = false;
1585    2
1586    2                  open file ( logical )  title( filename );
1587    2
1588    2
```

```
1455    1
1456    1
1457    1       %INCLUDE  'TRANSPORT.PLI';
1479    1
1480    1
1481    1
1482    1
1483    1
1484    1
1485    1       dcl  1   RECORD     ,
1486    1
1487    1
1488    1                2   Begin        char(30) var,
1489    1                2   distance     float bin(31),
1490    1                2   time         fixed bin(31),
1491    1                2   dist         char(15) var,
1492    1                2   Finish       char(30) var;
1493    1
1494    1       dcl  symbol char(3) var;
1495    1
1496    1       dcl position fixed bin(7);
1497    1
1498    1
1499    1
1500    1         dcl  alpha  char(30) var  init('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
1501    1
1502    1
1503    1
1504    1
1505    1       dcl  error    bit(1),
1506    1            true     bit(1) init('1'B),
1507    1            false    bit(1) init('0'B);
1508    1
1509    1
1510    1       dcl  more_records   bit(1);
1511    1
1512    1       dcl  reply      char(80) var,
1513    1            filename   char(12)  var;
1514    1
1515    1
1516    1       dcl  recnum      fixed bin(7),
1517    1            records     fixed bin(7),
1518    1            JJ          fixed bin(7);
1519    1
1520    1       dcl   logical  file   input   keyed sequential
1521    1
1522    1       environment( maximum_record_number(100),maximum_record_size(250));
1523    1
1524    1
1525    1       on  endfile(logical) more_records = false;
1526    1       more_records = true;
1527    1
1528    1       on undefinedfile  ( logical ) begin;
1529    2
1530    2          put skip(2) edit('The file',filename,'does not exist.')(A,X(1),A(12),X(1),A);
1531    2
```

```
1361    5                  Zones.Length(PATHS)= value;
1362    5                  Zones.Rate(PATHS)  = pointer;
1363    5                  Zones.D_name(PATHS)= dist;
1364    5                  Zones.End(PATHS)   = Endpoint;
1365    5
1366    5              end;
1367    4
1368    4
1369    4          end;
1370    3
1371    3                 PUT SKIP(3) EDIT('Would you like you Material Handling System Descriptions saved in a file?')
1372    3                 put skip edit('Reply ( Y/N ) : ')(A);
1373    3
1374    3                  err = true;
1375    3                    do while ( err );
1376    4
1377    4                       get edit(string1 ) ( A(80));
1378    4                     string1 = UCASE(string1);
1379    4                     if (string1 = 'Y' | string1 = 'N')then err = false;
1380    4
1381    4                      else put skip list('Invalid, re-enter : ');
1382    4                     end;
1383    3
1384    3           if( string1 = 'Y' ) then call MHS_SAVE;
1385    3
1386    3
1387    3
1388    3               END ;  /*  END OF NEW LOOP */
1389    2
1390    2
1391    2
1392    2       END; /* END OF WCD LOOP */
1393    2
1394    1
1395    1            DO WHILE ( old );
1396    1
1397    2             CALL  MHS_IN;
1398    2
1399    2            old = false;
1400    2              END; /* END OF OLD LOOP */
1401    2
1402    1
1403    1
1404    1          END   TRANS;
1405    1
1406
1407            MHS_IN  : proc;
1408    1       %include  'PAUSE.PLI';
1409    1
1418    1       %INCLUDE  'UCASE.PLI';
1419    1
1429    1       %INCLUDE  'FILETEST.PLI';
1430    1
1451    1
1452    1
1453    1
```

```
1                           float bin(31),float bin(31),float bin(31),
1                           float bin(31),float bin(31));
1
1       dcl   (pages,J,K,lower,upper,test) fixed bin(7);
1
1       dcl   (m1,v1,m2,v2,m3,v3 ) float bin(31);
1       dcl   reply  char(80) var;
1       dcl QHARD    ENTRY;
1
1
1       if( NUMQS <= 20 ) THEN pages = 1;
1       else if( NUMQS > 20  & NUMQS <= 40 ) THEN pages = 2;
1       else if( NUMQS > 40  & NUMQS <= 60 ) then pages = 3;
1       else if( NUMQS > 60  & NUMQS <= 80 ) THEN pages = 4;
1       else pages = 5;
1
1       lower = 1;
1       upper = MIN(20,NUMQS);
1       do J = 1 to pages;
2
2               call headings;
2               do K = lower to upper;
3
3                       call Calc_stat(3,K,m1,v1,m2,v2,m3,v3);
3                       put skip edit(K,m1,v1,m2,v2)(x(10),f(3),x(6),F(5,2),
3                             X(7),F(7,3),X(6),F(3),X(6),F(3));
3                       END;
2
2                                       lower = lower + 20;
2               test = upper + 20;
2               upper = MIN(test,NUMQS);
2               put skip edit('Hit a carriage return to continue.')(X(10),A);
2               get edit(reply)(A(80));
2               end;
1
1       put skip(10) edit('Would you like this report saved ? ')(x(10),A);
1       get edit(reply)(A(80));
1       reply = UCASE(reply);
1       if( reply = 'YES' ) then call QHARD;
1
1       headings : proc;
2
2       put skip (25);
2       put skip edit('Queue Summary Report')(X(28),A);
2       put skip edit('Queue','Average','Variance','MIN','MAX')
2                 (X(10),A,X(5),A,X(6),A,X(6),A,X(6),A);
2       PUT SKIP EDIT('Number','Size')(X(10),A,X(7),A);
2
2       end headings;
1
1       end QSTAT;

        /*****************************************/

        MHSTAT : proc;
1       %INCLUDE 'UCASE.PLI';
```

```
2     1       else if( works > 6 & works <= 12) then pages = 2;
3     1       else if( works > 12 & works <= 18) then pages = 3;
4     1       else if( works >18 & works <= 24) then pages = 4;
5     1       else pages = 5;
6     1
7     1
8     1       lower = 1;
9     1       upper = MIN(6,works);
0     1
1     1       do J = 1 to pages by 1;
2     2
3     2               call page_head;
4     2               do I = lower to upper;
5     3
6     3                       put file( WCUSTAT ) skip(2) edit("Work Center :  ",M_name(I)) (X(15),A,A);
7     3                       call print_head;
8     3                       call Calc_stat(i,I,m1,v1,m2,v2,m3,v3);
9     3                       put file( WCUSTAT ) skip edit(m1,v1,m2,v2,m3,v3)(X(7),f(5,2),x(3),f(7,3),
0     3                                                       x(6),f(5,2),x(3),f(7,3),
1     3                                                       x(7),f(5,2),x(3),f(7,3));
2     3                       end;
3     2
4     2               lower =lower + 6;
5     2               test = upper + 6;
6     2               upper = MIN(test,works);
7     2               end;
8     1
9     1
0     1
1     1
2     1       page_head : proc;
3     2
4     2       put file(WCUSTAT) PAGE edit("Work Center Utilization Report")(X(20),A);
5     2
6     2       end page_head;
7     1
8     1       print_head : proc;
9     2
0     2       PUT FILE( WCUSTAT ) skip(2) edit("Utilization","Blockage","% Down")
1     2                       (X(9),A(11),X(12),A(8),x(14),A(6));
2     2
3     2       PUT FILE( WCUSTAT ) skip edit("MEAN","VARIANCE","MEAN","VARIANCE","MEAN","VARIANCE")
4     2                 (X(8),A(4),X(3),A(8),X(6),A(4),X(4),A(8),X(6),A(4),X(4),A(8));
5     2
6     2       end print_head;
7     1
8     1       end
9     1               WCSTAT_HARD;
0
1
2
3               CSTAT : proc;
4     1
5     1       %INCLUDE "UCASE.PLI";
6     1       %INCLUDE "3.PLI";
7     1
8     1       dcl  Calc_stat  entry(fixed bin(7),fixed bin(7),float bin(31),
```

```
2               end;
1
1
1       err = true;
1       do while ( err ) ;
2
2               put skip(25) edit('Would like this report saved ?')(X(10),A);
2               put skip edit('Reply(YES/NO) :  ')(X(10),A);
2               get edit ( string) (A(80));
2               string = UCASE(string);
2               if( string = 'YES' | string = 'NO')then err = false;
2               else put skip edit('Invalid')(X(10),A);
2               end;
1
1       if(string = 'YES' ) then call WCSTAT_HARD;
1
1
1       page_head : proc;
2
2       put edit('Work Center Utilization Report')(X(20),A);
2
2       end page_head;
1
1       print_head : proc;
2
2       put skip(2) edit('Utilization','Blockage','% Down')
2                       (X(9),A(11),X(12),A(8),X(14),A(6));
2
2       put skip edit('MEAN','VARIANCE','MEAN','VARIANCE','MEAN','VARIANCE')
2                       (X(3),A(4),X(3),A(8),X(6),A(4),X(4),A(8),X(6),A(4),X(4),A(8));
2
2       end print_head;
1
1       end
1               WCSTAT;


        WCSTAT_HARD : proc;
1
1       %include 'MACHINE.PLI';%INCLUDE 'UCASE.PLI';
1
1       DCL( pages,lower,upper,test,J,I ) fixed bin(7);
1       dcl( m1,v1,m2,v2,m3,v3    )float bin(31);
1       dcl
1       string   char(80) var,  err     bit(1), true bit(1) init('1'B),
1       false    bit(1) init('0'B);
1
1       dcl
1               WCSTAT_HARD     entry,
1               Calc_stat       entry(fixed bin(7),fixed bin(7),float bin(31),
1                                       float bin(31),float bin(31),float bin(31),
1                                        float bin(31),float bin(31));
1
1
1       DCL  WCUSTAT  FILE STREAM OUTPUT PRINT;
1
1       if( works <= 6 ) then pages = 1;
```

```
2       PUT FILE(SHUTSTATS) PAGE SKIP(2) edit('Shuttle Utilization Report')(X(23),A);
2       PUT FILE(SHUTSTATS) SKIP(2) edit('Shuttle','%Utilization','%Blocked','Que','Type')
2                 (A,X(46),A,X(7),A,X(6),A,X(4),A);
2       PUT FILE(SHUTSTATS) SKIP EDIT('Name','Mean','Variance','Mean','Variance','Num')
2                 (A,X(28),A,X(1),A,X(2),A,X(1),A,x(4),A);
2       end page_head;
1
1       end SHTHARD;
        WCSTAT : proc;

1
1       %include 'MACHINE.PLI';%INCLUDE 'UCASE.PLI';
1
1       DCL( pages,lower,upper,test,J,I ) fixed bin(7);
1       dcl( m1,v1,m2,v2,m3,v3    )float bin(31);
1       dcl
1       string   char(80) var,  arr     bit(1), true bit(1) init('1'B),
1       false    bit(1) init('0'B);
1
1       dcl
1               WCSTAT_HARD     entry;
1               Calc_stat       entry(fixed bin(7),fixed bin(7),float bin(31),
1                                     float bin(31),float bin(31),float bin(31),
1                                      float bin(31),float bin(31));
1
1       if( works <= 3 ) then pages = 1;
1       else if( works > 3  & works <= 6 ) then pages = 2;
1       else if( works > 6  & works <= 9 ) then pages = 3;
1       else if( works > 9  & works <= 12) then pages = 4;
1       else if( works > 12 & works <= 15) then pages = 5;
1       else if( works > 15 & works <= 18) then pages = 6;
1       else if( works > 18 & works <= 21) then pages = 7;
1       else if( works > 21 & works <= 24) then pages = 8;
1       else pages = 9;
1
1       lower = 1;
1       upper = MIN(3,works);
1
1       do J = 1 to pages by 1;
2
2               put skip(25);
2               call page_head;
2               do I = lower to upper;
3
3                       put skip(3) edit('Work Center :  ',M_name(I)) (X(15),A,A);
3                       call print_head;
3                       call Calc_stat(1,I,m1,v1,m2,v2,m3,v3);
3                       put skip edit(m1,v1,m2,v2,m3,v3)(X(7),f(5,2),X(3),f(7,3),
3                                                        x(5),f(5,2),X(3),f(7,3),
3                                                        x(5),f(5,2),X(3),f(7,3));
3                       end;
2
2               lower =lower + 3;
2               test = uppar + 3;
2               upper = MIN(test,works);
2               put skip edit('Hit a carriage return to continue.')(X(5),A);
2               get edit(string)(a(80));
```

```
2       put skip(25) edit('Shuttle Utilization Report')(X(23),A);
2       put skip(2) edit('Shuttle','%Utilization','%Blocked','Que','Type')
2                    (A,X(28),A,X(7),A,X(4),A,X(2),A);
2       PUT SKIP EDIT('Name','Mean','Variance','Mean','Variance','Num')
2                  (A,X(27),A,X(1),A,X(2),A,X(1),A,X(4),A);
2       end page_head;
1
1       end SHTSTAT;
        SHTHARD : proc;

1       dcl  (pages,lower,upper,J,I,test)  fixed bin(7);
1       dcl  (m1,m2,m3,v1,v2,v3 )          float bin(31);
1       dcl  string                        char(80) var;
1       DCL  err  bit(1),true bit(1) init('1'B), false bit(1) init('0'B);
1       %include 'UCASE.PLI';


1       DCL  SHUTSTATS   FILE STREAM OUTPUT PRINT;

1       dcl  snthard   entry,
1            Calc_stat entry(fixed bin(7),fixed bin(7),float bin(31),
1                            float bin(31),float bin(31),float bin(31),
1                            float bin(31),float bin(31));


1       %include 'SHUTS.PLI';


1       OPEN FILE(SHUTSTATS);
1       IF( SHUTS <= 20 ) then pages = 1;
1       else if(SHUTS > 20 & SHUTS <= 40) then pages = 2;
1       else if(SHUTS > 40 & SHUTS <= 60) then pages = 3;
1       else if(SHUTS > 60 & SHUTS <= 80) then pages = 4;
1       else if(SHUTS > 80 )then pages = 5;

1       lower = 1;
1       upper = MIN(SHUTS,20);

1       do J =  1 to pages by 1;

2        CALL page_head;
2        do I = lower to upper ;

3               call Calc_stat(2,I,m1,v1,m2,v2,m3,v3);
3               PUT FILE(SHUTSTATS) SKIP edit(C_name(I),m1,v1,m2,v2,Que_num(I),IO(I))
3                           (A(30),X(2),F(4,2),X(2),F(5,3),X(4),F(4,2),X(2),
3                            F(5,3),X(6),F(3),X(6),A);
3                END;

2               lower = lower + 20;
2               test = upper + 20;
2               upper = MIN(test,SHUTS);
2               end;


1
1       page_head : proc;
```

```
1       dcl  (pages,lower,upper,J,I,test)  fixed bin(7);
1       dcl  (m1,m2,m3,v1,v2,v3 )          float bin(31);
1       dcl  string                        char(30) var;
1       DCL  err  bit(1),true bit(1) init('1'B), false bit(1) init('0'B);
1       %include 'UCASE.PLI';
1
1       dcl  SHTHARD   entry,
1            Calc_stat entry(fixed bin(7),fixed bin(7),float bin(31),
1                            float bin(31),float bin(31),float bin(31),
1                            float bin(31),float bin(31));
1
1
1       %include 'SHUTS.PLI';
1
1       IF( SHUTS <= 20 ) then pages = 1;
1       else if(SHUTS > 20 & SHUTS <= 40) then pages = 2;
1       else if(SHUTS > 40 & SHUTS <= 60) then pages = 3;
1       else if(SHUTS > 60 & SHUTS <= 80) then pages = 4;
1       else if(SHUTS > 80 )then pages = 5;
1
1       lower = 1;
1       upper = MIN(SHUTS,20);
1
1       do J =  1 to pages by 1;
2
2        CALL page_head;
2        do I = lower to upper ;
3
3               call Calc_stat(2,I,m1,v1,m2,v2,m3,v3);
3               put skip edit(C_name(I),m1,v1,m2,v2,Que_num(I),ID(I))
3                            (A(30),X(2),F(4,2),X(2),F(5,3),X(3),F(4,2),X(2),
3                             F(5,3),X(2),F(3),X(2),A);
3                END;
2
2               lower = lower + 20;
2               test = upper + 20;
2               upper = MIN(test,SHUTS);
2               put skip edit('Hit a carriage return to continue.')(X(5),A);
2               get edit(string)(A(80));
2               end;
1
1       err = true;
1       do while ( err ) ;
2
2               put skip(25) edit('Would like this report saved ?')(X(10),A);
2               put skip edit('Reply(YES/NO) :  ')(X(10),A);
2               get edit ( string) (A(80));
2               string = UCASE(string);
2               if( string = 'YES' | string = 'NO')then err = false;
2               else put skip edit('Invalid')(X(10),A);
2               end;
1
1       if(string = 'YES' ) then call SHTHARD;
1
1
1       page_head : proc;
2
```

```
1       OUTPUT : proc;
1
1       dcl  reply  char(80) var;
1
1       dcl     WCSTAT  ENTRY,
1               SHTSTAT ENTRY,
1               MHSTAT  ENTRY,
1               WIPSTAT ENTRY,
1               QSTAT   ENTRY,
1               DOWNSTAT        ENTRY;
1
1
1       DO WHILE( reply ^=  "7" );
2       CALL out_menu;
2
2       reply = "";
2       put skip(2) edit("Which report would like to see ?  ")(X(15),A);
2       get edit(reply)(A(80));
2
2       if( reply = "1") then call WCSTAT;
2       else if(reply ="2")than call    SHTSTAT;
2       else if(reply = "3")then call   MHSTAT;
2       else if( reply = "4")then call  WIPSTAT;
2       else if( reply = "5")then call  QSTAT;
2       else if( reply = "6")then call  DOWNSTAT;
2       else if( reply ^= "7")then
2
2               put skip(2) edit("Invalid reply.")(X(15),A);
2
2       end;
1
1       out_menu : proc;
2
2       put skip(25);
2       put skip edit("                                                              ")(X(9),A);
2       put skip edit("┌──────────────────────────────────────────────────────────┐  ")(X(9),A);
2       put skip edit("│                    SYSTEM REPORTING                      │  ")(X(9),A);
2       put skip edit("│                                                          │  ")(X(9),A);
2       put skip edit("│      Option                    Report                    │  ")(X(9),A);
2       put skip edit("│        1.    Work Center Utilization Report              │  ")(X(9),A);
2       put skip edit("│        2.    Shuttle Utilization Report                  │  ")(X(9),A);
2       put skip edit("│        3.    Material Handling Utilization Report        │  ")(X(9),A);
2       put skip edit("│        4.    Work in Process and Production Report       │  ")(X(9),A);
2       put skip edit("│        5.    Queue Summary Report                        │  ")(X(9),A);
2       put skip edit("│        6.    Breakdown Summary Report                    │  ")(X(9),A);
2       put skip edit("│        7.    Exit                                        │  ")(X(9),A);
2       put skip edit("│                                                          │  ")(X(9),A);
2       put skip edit("└──────────────────────────────────────────────────────────┘  ")(X(9),A);
2
2       end out_menu;
1
1       end OUTPUT;

        SHTSTAT : proc;
1
```

```
1876 | 1      /*************************************************************/
1877   1
1878   1      instructions : proc;
1879   2
1880   2      put skip(5) edit('++++  Filing System Instructions  ++++') ( X(21),A);
1881   2      put skip(3) edit('In the filing system you will be asked to enter') (X(15),A);
1882   2      put skip edit('a file name where your model statements can be')(X(10),A);
1883   2      put skip edit('saved.  The file name can consist of from 1 to 8')(X(10),A);
1884   2      put skip edit('letters or numbers.  However, the first character') (X(10),A);
1885   2      put skip edit('must be a letter.  Examples of valid file names')(X(10),A);
1886   2      put skip edit('are show below.') ( X(10),A);
1887   2      put skip(2) edit('Valid File Names')(X(29),A);
1888   2      put skip(2) edit('USERFILE ')(X(33),A);
1889   2      put skip    edit('MACHINE2 ')(X(33),A);
1890   2      put skip    edit('FILE2326 ')(X(33),A);
1891   2      put skip    edit('F1234567' )(X(33),A);
1892   2
1893   2
1894   2      END instructions;
1895   1
1896   1
1897   1      END  MHS_SAVE;
```

APPENDIX E

EXAMPLE MODEL I

```
!!!!  SHUTTLE  DESCRIPTIONS  !!!!

Do you wish to use NEW or EXISTING Shuttle descriptions?
Reply ( NEW/OLD ):  NEW


Do you need to see the instruction set? ( YES/NO ):  NO

Shuttle name:  ROBOT_1

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,1,LOAD,O,NORMAL



Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  25

Enter the mean for the distribution :  .63

Enter a standard deviation that is greater than 0.0 :  .04


Shuttle name:  ROBOT_1

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,1,CNCLATHE,I,NORMAL

Queue number   1 already has a capacity of     25
Would you like to change that capacity? (Y/N): N

Enter the mean for the distribution :  .55

Enter a standard deviation that is greater than 0.0 :  .035

Shuttle name:  TRANSFER1

Capacity , Q Rank , Q number , Work Center , I/O , Distributio

>1,FIFO,2,CNCLATHE,O,T


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10

Enter the Upper and Lower limits for the Triangular distribution: .5,.3

Enter the Mode of the distribution:  .42


Shuttle name:  TRANSFER1

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,2,DNC_MILL,I,U

Queue number   2 already has a capacity of     10
Would you like to change that capacity? (Y/N): N

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  .5,.35

Shuttle name:  INDEXER1

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,3,DNC_MILL,O,U


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  .45,.3

Shuttle name:  INDEXER1

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,3,NC_DRILL1,I,N

Queue number   3 already has a capacity of     10
Would you like to change that capacity? (Y/N): N

Enter the mean for the distribution :  .38

Enter a standard deviation that is greater than 0.0 :  .025

Shuttle name:  ROBOT_2

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,4,NC_DRILL1,O,N
```

```
Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10

Enter the mean for the distribution :  .375

Enter a standard deviation that is greater than 0.0 :  .027

Shuttle name:   ROBOT_2

Capacity , Q Rank , Q number , Work Center , I/O , Distribution

>1,FIFO,4,INSPECTION,I,N

Queue number   4 already has a capacity of    10
Would you like to change that capacity? (Y/N): N

Enter the mean for the distribution :  .39

Enter a standard deviation that is greater than 0.0 :  .024

Shuttle name:   QUIT

Would you like your descriptions saved in a file:( Y/N )?   N

                          xxxx  WORK CENTER DESCRIPTIONS  xxxx

     Do you wish to use NEW or EXISTING Work Center Descriptions ?
     Reply ( NEW/OLD ):  NEW


     Do you need to see the instruction set ?(YES/NO)  NO

          How many work centers are there in your system ?5

     Work Center  1      LOAD,1,L

     Work Center  2      CNCLATHE,1,UC

     Work Center  3      DNC_MILL,1,UC

     Work Center  4      NC_DRILL1,2,UC

     Work Center  5      INSPECTION,1,UL


     Would you like the Work Center descriptions saved in a file ? ( Y/N ):  N
```

Work Center Utilization Report

Work Center: LOAD

Utilization		Blockage		% Down	
Mean	Variance	Mean	Variance	Mean	Variance
0.00	0.000	0.00	0.000	0.00	0.000

Work Center: CNCLATHE

Utilization		Blockage		% Down	
Mean	Variance	Mean	Variance	Mean	Variance
0.89	0.063	0.00	0.000	0.00	0.000

Work Center: DNC_MILL

Utilization		Blockage		% Down	
Mean	Variance	Mean	Variance	Mean	Variance
0.46	0.248	0.00	0.000	0.00	0.000

Work Center: NC_DRILL1

Utilization		Blockage		% Down	
Mean	Variance	Mean	Variance	Mean	Variance
0.77	0.325	0.00	0.000	0.08	0.250

Work Center: INSPECTION

Utilization		Blockage		% Down	
Mean	Variance	Mean	Variance	Mean	Variance
0.27	0.198	0.00	0.000	0.00	0.000

Work in Progress and Production Report

Work in Process

Average number of parts in the system	=	8.50
Variance	=	13.24

System Production

Average Time in the System	=	50.732
Variance	=	719.308
Part Production per unit time	=	0.019
Variance	=	0.000636

Production by Part Type

Part name	Avg Tis	Variance	Number
PART_1	44.166	0.001	40.000
PART_2	56.839	0.002	43.000

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	4.12	8.687	0	11
2	0.00	0.000	0	0
3	0.00	0.000	0	0
4	0.00	0.000	0	0

APPENDIX F

EXAMPLE MODEL II

```
                    $$$$  Part Descriptions  $$$$


This is the beginning of the part input sequence

        Enter a part name or 'QUIT' :  PART_1

        How many operations are there for this part?4


Enter the distribution name for processing time associated with each operation
and the part dispatching rule.  The input format is as follows:

Operation 1 --   distribution_name,dispatching_rule

Operation  1  N,SNIQ

Enter the mean for the distribution :  6.5

Enter a standard deviation that is greater than 0.0 :  .32


   Next you will enter from 1 to 10 work center names
   where operation 1 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >CNCLATHE

 >STOP

Operation  2  T,SNIQ

Enter the Upper and Lower limits for the Triangular distribution: 3.0,2.2

Enter the Mode of the distribution:  2.45


   Next you will enter from 1 to 10 work center names
   where operation 2 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >DNC_MILL

 >STOP
```

```
Operation  3  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2.1,1.73


   Next you will enter from 1 to 10 work center names
   where operation 3 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >MC_DRILL1

 >STOP

Operation  4  N,SNIQ

Enter the mean for the distribution :  1.5

Enter a standard deviation that is greater than 0.0 :  .25


   Next you will enter from 1 to 10 work center names
   where operation 4 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >INSPECTION

 >STOP


  The next input section involves specifying the sequencing
  of operations.  When you are given  an '*' as a prompt you
  can enter the operation numbers in the order that they
  are to be performed similar to the following example :

    Assume there are 5 different operations, the input sequence
    would look like the following :

      Operation Sequence
      *   4
      *   1
      *   5
      *   2
      *   3


      Operation Sequencing

*1

*2
```

```
x3
x4

Enter a part name or 'QUIT'  :  PART_2

How many operations are there for this part?4


Enter the distribution name for processing time associated with each operation
and the part dispatching rule.  The input format is as follows:

Operation 1     distribution_name,dispatching_rule

Operation  1  N,SNIQ

Enter the mean for the distribution :  5.5

Enter a standard deviation that is greater than 0.0 :  .34


   Next you will enter from 1 to 10 work center names
   where operation 1 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.


 >CNCLATHE

 >STOP

Operation  2  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2.75,2.43


   Next you will enter from 1 to 10 work center names
   where operation 2 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.


 >DNC_MILL

 >STOP

Operation  3  U,SBBQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2.1,1.8
```

```
   Next you will enter from 1 to 10 work center names
   where operation 3 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >NC_DRILL1

 >STOP

Operation  4  N,SNIQ

Enter the mean for the distribution :  1.65

Enter a standard deviation that is greater than 0.0 :  .375


   Next you will enter from 1 to 10 work center names
   where operation 4 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >INSPECTION

 >STOP


  The next input section involves specifying the sequencing
  of operations.  When you are given  an 'X' as a prompt you
  can enter the operation numbers in the order that they
  are to be performed similar to the following example :

   Assume there are 5 different operations, the input sequence
   would look like the following :

     Operation Sequence
     X   4
     X   1
     X   5
     X   2
     X   3


     Operation Sequenceing

X1

X2

X3

X4
```

```
  Enter a part name or 'QUIT'  :  CHANGE_OVER

          How many operations are there for this part?4


Enter the distribution name for processing time associated with each operation
and the part dispatching rule.  The input format is as follows:

Operation 1      distribution_name,dispatching_rule

Operation  1  N,SNIQ

Enter the mean for the distribution :  3.5

Enter a standard deviation that is greater than 0.0 :  .2


   Next you will enter from 1 to 10 work center names
   where operation 1 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >CNCLATHE

 >STOP

Operation  2  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2.5,2.0


   Next you will enter from 1 to 10 work center names
   where operation 2 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >DNC_MILL

 >STOP
```

```
Operation  3  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  3.75,3.10


   Next you will enter from 1 to 10 work center names
   where operation 3 can be performed.
   The prompt for input is ">" and you can type
   "STOP" to terminate the input sequence.

 >MC_DRILL1

 >STOP

Operation  4  N,SNIQ

Enter the mean for the distribution :  1.0

Enter a standard deviation that is greater than 0.0 :  .1


   Next you will enter from 1 to 10 work center names
   where operation 4 can be performed.
   The prompt for input is ">" and you can type
   "STOP" to terminate the input sequence.

 >INSPECTION

 >STOP
```

The next input section involves specifying the sequencing of operations. When you are given an "*" as a prompt you can enter the operation numbers in the order that they are to be performed similar to the following example :

Assume there are 5 different operations, the input sequence would look like the following :

```
  Operation Sequence
  *   4
  *   1
  *   5
  *   2
  *   3
```

```
   Operation Sequenceing

*1

*2

*3

*4


  Enter a part name or "QUIT"  :  QUIT
```

```
                    >>>>  Production Scheduling  <<<<


     This is the beginning of the Production Schedule input sequence.

     In this section you will be asked to input three items
     to describe the production schedule.  The three items are the
     part name, the lot size, and the frequency for loading individual
     parts into the system.  The production lots should be entered in the
     sequence you would like them to be processed.

     Enter a part name or 'QUIT' :  PART_1


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  20,N

Enter the mean for the distribution :  8.0

Enter a standard deviation that is greater than 0.0 :  .9

     Enter a part name or 'QUIT' :   CHANGE_OVER


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  1,N

Enter the mean for the distribution :  10.0

Enter a standard deviation that is greater than 0.0 :  .5

     Enter a part name or 'QUIT' :   PART_2


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  30,N

Enter the mean for the distribution :  7.5

Enter a standard deviation that is greater than 0.0 :  .85

     Enter a part name or 'QUIT' :   CHANGE_OVER


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  1,N

Enter the mean for the distribution :  9.0

Enter a standard deviation that is greater than 0.0 :  .75

     Enter a part name or 'QUIT' :   PART_1


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  40,N

Enter the mean for the distribution :  8.0

Enter a standard deviation that is greater than 0.0 :  .9

     Enter a part name or 'QUIT' :   QUIT
```

```
Enter the starting and stopping times for
the simulation seperating the values by commas.
Times =    0,480


Do you wish to use the TRACE option ? (Yes/No) NO

Do you wish to have statistics cleared
after a certain clock time ? (Yes/No) NO


What is the TOTAL number of parts
allowed in the system at one time?
Max parts =  24


What is the Total number of CARRIERS
available in the Material Handling System?
Max CARRIERS =    0
```

Work Center Utilization Report

Work Center : LOAD

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.00	0.000	0.00	0.000	0.00	0.000

Work Center : CNCLATHE

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.75	0.185	0.00	0.000	0.00	0.000

Work Center : DNC_MILL

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.32	0.219	0.00	0.000	0.00	0.000

Work Center Utilization Report

Work Center : NC_DRILL1

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.24	0.185	0.00	0.000	0.00	0.000

Work Center : INSPECTION

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.20	0.161	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Shuttle Utilization Report

Shuttle Name	%Utilization Mean	%Utilization Variance	%Blocked Mean	%Blocked Variance	Que Num	Type
ROBOT_1	0.06	0.063	0.00	0.000	1	O
ROBOT_1	0.01	0.012	0.00	0.000	1	I
TRANSFER1	0.13	0.118	0.00	0.000	2	O
TRANSFER1	0.00	0.000	0.00	0.000	2	I
INDEXER1	0.04	0.044	0.00	0.000	3	O
INDEXER1	0.00	0.000	0.00	0.000	3	I
ROBOT_2	0.04	0.044	0.00	0.000	4	O
ROBOT_2	0.00	0.000	0.00	0.000	4	I

Hit a carriage return to continue.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 1.83

Variance = 0.22

System Production

Average Time in the System = 14.661

Variance = 0.681

Part Production per unit time = 0.068

Variance = 0.000045

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	0.01	0.012	0	1
2	0.00	0.000	0	0
3	0.00	0.000	0	0
4	0.00	0.000	0	0

Hit a carriage return to continue.

Production by Part Type

Part name	Avg Tis	Var	Number
PART_1	15.146	2.403	28.000
PART_2	14.351	2.923	30.000
CHANGE_OVER	12.524	5.129	2.000

Hit a carriage return to continue.

APPENDIX G

EXAMPLE MODEL III

```
Do you need to see the instruction set ?(YES/NO)  NO
     How many work centers are there in your system ?6
Work Center  1     LOAD,1,L
Work Center  2     DRILL1,1,WC
Work Center  3     MILL1,1,WC
Work Center  4     DRILL2,1,WC
Work Center  5     DEBUR,1,WC
Work Center  6     INSP,1,WL

Would you like the Work Center descriptions saved in a file ? ( Y/N ):  Y
```

```
                    ****  Part Descriptions  ****

This is the beginning of the part input sequence

     Enter a part name or 'QUIT' :  PART1
```

```
How many operations are there for this part?5


Enter the distribution name for processing time associated with each operation
and the part dispatching rule.  The input format is as follows:

Operation 1      distribution_name,dispatching_rule

Operation  1  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2,1.5


   Next you will enter from 1 to 10 work center names
   where operation 1 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >DRILL1

 >STOP

Operation  2  N,SNIQ

Enter the mean for the distribution :  2.75

Enter a standard deviation that is greater than 0.0 :  .3


   Next you will enter from 1 to 10 work center names
   where operation 2 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >MILL1

 >STOP
```

```
Operation  3  U,SNIQ

Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  2.25,1.85


   Next you will enter from 1 to 10 work center names
   where operation 3 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >DRILL2

 >STOP

Operation  4  N,SNIQ

Enter the mean for the distribution :  1.75

Enter a standard deviation that is greater than 0.0 :  .2


   Next you will enter from 1 to 10 work center names
   where operation 4 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >DEBUR

 >STOP

Operation  5  N,SNIQ

Enter the mean for the distribution :  1.5

Enter a standard deviation that is greater than 0.0 :  .15


   Next you will enter from 1 to 10 work center names
   where operation 5 can be performed.
   The prompt for input is '>' and you can type
   'STOP' to terminate the input sequence.

 >INSP

 >STOP
```

The next input section involves specifying the sequencing of operations. When you are given an 'x' as a prompt you can enter the operation numbers in the order that they are to be performed similar to the following example :

Assume there are 5 different operations, the input sequence would look like the following :

```
  Operation Sequence
  x   4
  x   1
  x   5
  x   2
  x   3


  Operation Sequenceing

x1

x2

x3

x4

x5

Enter a part name or 'QUIT'  :  QUIT
```

```
Would you like to make any changes? ( Y/N ):  N



          >>>>  Production Scheduling  <<<<


     This is the beginning of the Production Schedule input sequence.

     In this section you will be asked to input three items
     to describe the production schedule.  The three items are the
     part name, the lot size, and the frequency for loading individual
     parts into the system.  The production lots should be entered in the
     sequence you would like them to be processed.

     Enter a part name or "QUIT" :  PART1


     Enter the "lot size" and the "input frequency"
     seperating the items by a comma.
     :  500,N
Enter the mean for the distribution :  4.5
Enter a standard deviation that is greater than 0.0 :  .5
     Enter a part name or "QUIT" :   QUIT
```

```
          <<<<  Material Handling System Description  >>>>
How many paths emanate from station LOAD  1
Enter the end point, the path length, and the travel rate for path   1
:   DRILL1,50,N
Enter the mean for the distribution :  100
Enter a standard deviation that is greater than 0.0 :  5
How many paths emanate from station DRILL1  1
Enter the end point, the path length, and the travel rate for path   1
:   MILL1,20,N
Enter the mean for the distribution :  100
Enter a standard deviation that is greater than 0.0 :  5
How many paths emanate from station MILL1  1
Enter the end point, the path length, and the travel rate for path   1
:   DRILL2,20,N
Enter the mean for the distribution :  100
Enter a standard deviation that is greater than 0.0 :  5
How many paths emanate from station DRILL2  1
Enter the end point, the path length, and the travel rate for path   1
:   DEBUR,20,N
```

```
Enter the mean for the distribution :  100
Enter a standard deviation that is greater than 0.0 :  5
How many paths emanate from station DEBUR  1
Enter the end point, the path length, and the travel rate for path   1
*   INSP,50,N
Enter the mean for the distribution :  100
Enter a standard deviation that is greater than 0.0 :  5
How many paths emanate from station INSP  0

Would you like you Material Handling System Descriptions saved in a file?
Reply ( Y/N ) :  N
```

```
                                                    Enter the value for the constant:   .5
Shuttle name:  EX1
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,1,LOAD,O,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :
Enter the value for the constant:   .5
Shuttle name:   EX2
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,2,DRILL1,I,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX3
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,3,DRILL1,O,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX4
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,4,MILL1,I,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX5
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,5,MILL1,O,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
```

```
Shuttle name:   EX6
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,6,DRILL2,I,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX7
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,7,DRILL2,O,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX8
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,8,DEBUR,I,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX9
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,9,DEBUR,O,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX10
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,10,INSP,I,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   QUIT

Would you like your descriptions saved in a file:( Y/N )?   N
```

Work Center Utilization Report

Work Center : DRILL2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.45	0.247	0.00	0.000	0.00	0.000

Work Center : DEBUR

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.38	0.236	0.00	0.000	0.00	0.000

Work Center : INSP

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.32	0.219	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

APPENDIX H

EXAMPLE MODEL IV

```
                    >>>>  Production Scheduling  <<<<


     This is the beginning of the Production Schedule input sequence.

     In this section you will be asked to input three items
     to describe the production schedule.  The three items are the
     part name, the lot size, and the frequency for loading individual
     parts into the system.  The production lots should be entered in the
     sequence you would like them to be processed.

     Enter a part name or 'QUIT' :  PART_1


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  20,N

Enter the mean for the distribution :  8.0

Enter a standard deviation that is greater than 0.0 :  .9

     Enter a part name or 'QUIT' :   CHANGE_OVER


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  1,N

Enter the mean for the distribution :  10.0

Enter a standard deviation that is greater than 0.0 :  .5

     Enter a part name or 'QUIT' :   PART_2


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  30,N

Enter the mean for the distribution :  7.5

Enter a standard deviation that is greater than 0.0 :  .85

     Enter a part name or 'QUIT' :   CHANGE_OVER


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  1,N

Enter the mean for the distribution :  9.0

Enter a standard deviation that is greater than 0.0 :  .75

     Enter a part name or 'QUIT' :   PART_1


     Enter the 'lot size' and the 'input frequency'
     seperating the items by a comma.
     :  40,N

Enter the mean for the distribution :  8.0

Enter a standard deviation that is greater than 0.0 :  .9

     Enter a part name or 'QUIT' :   QUIT
```

```
Enter the mean for the distribution :  .55
Enter a standard deviation that is greater than 0.0 :  .062
Shuttle name:   ROBOT2
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,8,CNC_LATHE,I,N
Queue number   8 already has a capacity of     10
Would you like to change that capacity? (Y/N): N
Enter the mean for the distribution :  .47
Enter a standard deviation that is greater than 0.0 :  .032
Shuttle name:   EX7
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,9,CNC_LATHE,O,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10
Enter the value for the constant:   .46
Shuttle name:   EX8
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,10,DEBUR,I,U


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10
Enter the Upper and Lower limits for the Uniform
distribution; seperating the values by a comma(,):  .25,.166
Shuttle name:   EX9
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,10,DEBUR,O,C
Queue number  10 already has a capacity of     10
Would you like to change that capacity? (Y/N): N
Enter the value for the constant:   .45
Shuttle name:   EX10
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,12,INSP,I,C
Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10
Enter the value for the constant:   .5
Shuttle name:   QUIT

Would you like your descriptions saved in a file:( Y/N )?   N
```

```
Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX6
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,6,DRILL2,O,C


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   ROBOT1
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,7,DRILL2,I,N


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10
Enter the mean for the distribution :  .5
Enter a standard deviation that is greater than 0.0 :  .035
Shuttle name:   EX6
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,6,DNC_MILL2,I,C

Queue number  6 has a ranking criteria of SPT
You specified the new criteria as FIFO
Would you like the 'NEW' or 'OLD' ranking criteria ?
Enter ?(NEW/OLD) ?  NEW

Queue number   6 already has a capacity of      5
Would you like to change that capacity? (Y/N): N

Enter the value for the constant:   .5
Shuttle name:   ROBOT2
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,8,DNC_MILL2,O,N


Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  10
```

```
Shuttle name:  EX1
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,1,LOAD,O,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :
Enter the value for the constant:   .5
Shuttle name:   EX2
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,2,DRILL1,I,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX3
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,3,DRILL1,O,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX4
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,SPT,4,MILL1,I,C

Enter the MAXIMUM QUEUE SIZE ( Return = infinite) :  5
Enter the value for the constant:   .5
Shuttle name:   EX5
Capacity , Q Rank , Q number , Work Center , I/O , Distribution
>1,FIFO,5,MILL1,O,C
```

Work Center Utilization Report

Work Center : DRILL2

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.45	0.247	0.00	0.000	0.00	0.000

Work Center : DEBUR

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.38	0.236	0.00	0.000	0.00	0.000

Work Center : INSP

Utilization		Blockage		% Down	
MEAN	VARIANCE	MEAN	VARIANCE	MEAN	VARIANCE
0.32	0.219	0.00	0.000	0.00	0.000

Hit a carriage return to continue.

Work in Process and Production Report

Work in Process

Average number of parts in the system = 1.83

Variance = 0.22

System Production

Average Time in the System = 14.661

Variance = 0.681

Part Production per unit time = 0.068

Variance = 0.000045

Hit a carriage return to continue.

Shuttle Utilization Report

Shuttle Name	%Utilization Mean	%Utilization Variance	%Blocked Mean	%Blocked Variance	Que Num	Type
ROBOT_1	0.06	0.063	0.00	0.000	1	O
ROBOT_1	0.01	0.012	0.00	0.000	1	I
TRANSFER1	0.13	0.118	0.00	0.000	2	O
TRANSFER1	0.00	0.000	0.00	0.000	2	I
INDEXER1	0.04	0.044	0.00	0.000	3	O
INDEXER1	0.00	0.000	0.00	0.000	3	I
ROBOT_2	0.04	0.044	0.00	0.000	4	O
ROBOT_2	0.00	0.000	0.00	0.000	4	I

Hit a carriage return to continue.

Queue Summary Report

Queue Number	Average Size	Variance	MIN	MAX
1	0.01	0.012	0	1
2	0.00	0.000	0	0
3	0.00	0.000	0	0
4	0.00	0.000	0	0

Hit a carriage return to continue.

VITA 4/

William Henry Remy, III

Candidate for the Degree or

Master of Science

Thesis: DEVELOPING A SIMULATION LANGUAGE FOR FLEXIBLE MANUFACTURING SYSTEMS

Major Field: Industrial Engineering and Management

Biograhical:

Personal Data: Born in Kansas City, Missouri, April 4, 1959, the son of Charles F. and Judith C. Lyman. Married Janis L. Lacy on December 31, 1982.

Education: Graduated from Mission San Jose High School, Fremont, California, June, 1977; received Bachelor of Science degree in Industrial Engineering from Oklahoma State University in July, 1982; completed requirements for the Master of Science degree at Oklahoma State University in May, 1984.

Professional Experience: Graduate Assistant, School or Industrial Engineering and Management, Oklahoma State University, August, 1982 to December, 1983; Member of the Institute of Industrial Engineers and the Operations Research Society of America; Manufacturing Technology Engineer, General Dynamics Corporation, Fort Worth Division, Fort Worth, Texas, February 1984 to present.