Computer Science and Systems Analysis

Computer Science and Systems Analysis

Technical Reports

Miami University *Year* 1996

A Simulation Tool for the Manufacturing Engineering Department's Flexible Manufacturing Cell

Don Anderson
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1996-002

A Simulation Tool for the Manufacturing Engineering Department's Flexible Manufacturing Cell

Don Anderson

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

A Simulation Tool for the Manufacturing Engineering Department's Flexible Manufacturing Cell

by
Don Anderson
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #96-002 April, 1996

Systems Analysis Master's Project:

A Simulation Tool for the Manufacturing Engineering Department's Flexible Manufacturing Cell.

The flexible manufacturing cell in the manufacturing engineering department's computer integrated manufacturing systems lab is used to study manufacturing processes. A loop conveyor, an automated storage and retrieval system, robots and a lathe station are configured to exemplify a manufacturing process. This software was developed to allow the user to alter, modify and/or expand on the manufacturing process represented, and to conduct simulation studies relative to it. It presumes that the basic structure of the cell is to remain intact, while the type of and number of stations, and the behavior of the conveyor at each of the various stations, can be modified.

Donald W. Anderson

April 23, 1996

Table of Contents:

1.0 Introduction

> "Given the rapidly emerging and highly competitive global market-place, industry is constantly seeking methods of enhancing their competitive postures. For manufacturers, the certainty that a product, process, or machine will function exactly as intended prior to its actual implementation is an enormous potential advantage in terms of cost, reliability and lead time to market. Lengthy product testing and prototype construction is prohibitively expensive. Computer simulation offers an alternative to traditional design and testing methodology. What is needed is a more efficient process of concept and design testing.
>
> Formerly reserved for large-scale military and scientific projects, computing power has reached a level of exceptional performance coupled with such low costs that complex machining processes, facility layouts and material performance characteristics simulations can be conducted by almost any commercial enterprise in the United States. The impressive accuracy obtained by these simulations and the inherent cost savings make computer simulation, or computational modeling as it is also known, a force to be reckoned with in the coming years."
>
> -Nwoke and Nelson [1993]

Engineers, technologists and managers of manufacturing concerns have long sought tools and techniques to assist with the consequential, costly, and often troublesome analysis and design of manufacturing systems. Whether designing a new manufacturing system or seeking to modify and improve an existing system, computer simulation can be a desirable, useful and cost-effective technique.

Empirical methods and mathematical models can sometimes provide the required information, and actually have the advantage in some cases of finding an optimal solution. But as systems become large and more complicated, these methods become less desirable. Modifying and experimenting with the actual system can be very costly and time consuming, and most mathematical models require simplifying assumptions which

can limit the model's effectiveness. In some cases, computer simulation is the only viable means of conducting a comparative performance analysis.

It is not my intention here to develop a detailed comparison of various analytical techniques. For such a comparison, the interested reader is referred to Winston [1991] or Hoover and Perry [1990]. My purpose here is to loosely define computer simulation and explore how it can be exemplified and utilized with respect to the Flexible Manufacturing Cell (FMC) in the Manufacturing Engineering Department of Miami University's Computer Integrated Manufacturing Systems (CIMS) lab.

Musil and Akbay [1989] define simulation as ". . . the process of designing a mathematical-logical model of a system and performing experiments with this model on a computer." The reasons or rationales for conducting simulation studies are varied and diverse. In a manufacturing setting, one might be interested in evaluating alternative configurations for a group of machines, considering alternative material handling equipment, or determining the required size of or location of storage facilities. Perhaps a decision as to how to best increase the capacity of a certain manufacturing cell, or how to alter the cell so that the output is more constant, or faster, or less expensive. Maybe defective parts are causing problems downstream, or an assembly line or flexible manufacturing system (FMS) cell is to be evaluated to determine if it can accommodate an additional item or process.

Law and Haider [1989] describe two major types of manufacturing analysis for which simulation is used. In a *high-level analysis* the system is modeled at an aggregate level and details of the control logic are not included. High-level analyses are often performed in the initial phases of system design, when detailed system information is not yet available. Typical objectives of a high-level analysis would be to determine the required number of machines and material handling equipment, evaluating the effect of a change in product mix, and determining the storage requirements for work in process.

Detailed analyses, on the other hand, are performed on existing or proposed systems, to fine tune or optimize the system's performance. In a detailed analysis,

variations in control logic and processing strategies are evaluated to determine optimum approaches for material handling equipment and servers, and to determine the most suitable product mixes and queuing priorities. At this level a precise system description is needed.

The particular topics or areas of study are determined largely by the specific manufacturing task and manufacturing system being considered, as well as specific time and budgetary constraints. This paper will examine various simulation techniques and methodologies, and survey the simulation software that is currently available. Desirable features of simulation software will be identified which are particularly applicable to the Flexible Manufacturing Cell (FMC) in the Manufacturing Engineering Department's Computer-Integrated Manufacturing Systems (CIMS) lab. Requirements for the simulation tool will be determined and alternative designs developed and evaluated. Specifications for the proposed system will then be presented and the software developed. Validation of the software will be accomplished via comparison with an existing simulation software system.

2.0 The Manufacturing Engineering Department's Flexible Manufacturing Cell

The Manufacturing Engineering Department's Flexible Manufacturing Cell (FMC) is used primarily to demonstrate and exemplify a manufacturing process. The cell consists of an automatic storage and retrieval system, two RM-501 Mitsubishi robots, a Span Tech XL loop conveyor, and an Emco Maier CNC lathe (see figure 1). Currently, the cell is automated in that the storage and retrieval system and the lathe station are each individually PC controlled. For the storage and retrieval system, an operator specifies using a PC which storage location's contents are to be loaded onto the system, and which storage location should house a piece being removed from the system. The PC that governs the lathe is programmed by the user to produce parts with specified dimensions.

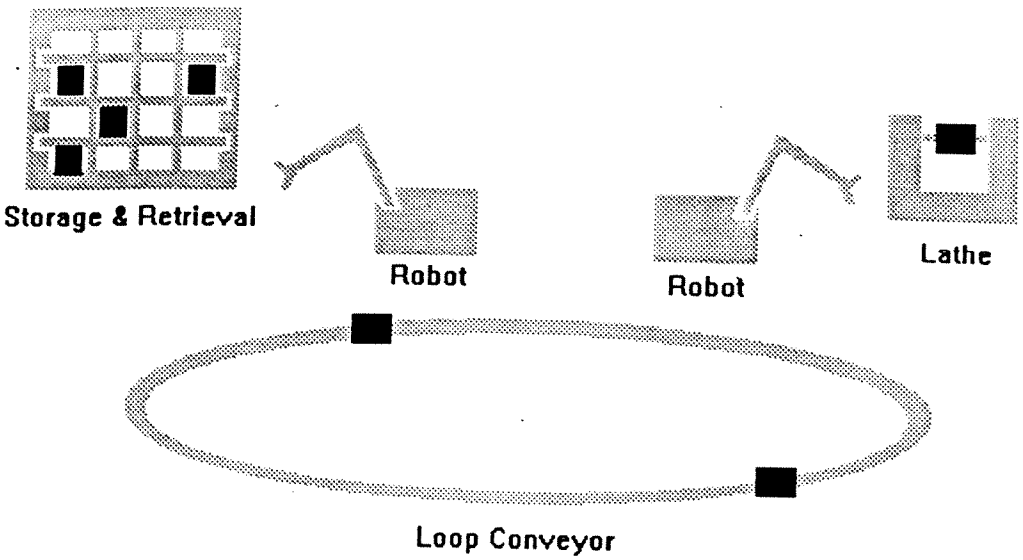


figure 1 the manufacturing engineering department's FMC

Parts are transported along the conveyor on pallets. When a pallet arrives at the work station (lathe), the part is unloaded from the pallet, loaded onto the machine, and the pallet remains on the conveyor. Two features of the system are unusual, and noteworthy: first, the conveyor stops and waits while the part is processed on the lathe, and second, pallets must be placed onto and removed from the conveyor by hand.

Dr. Ettouney (Associate Professor of Manufacturing Engineering) recently proposed changing the configuration of the manufacturing cell. He hopes to extend the conveyor to allow for additional service stations so that the cell can be used to demonstrate more complicated manufacturing processes. While considering alternative configurations for the new manufacturing cell, Dr. Ettouney realized that his students could benefit from the exercise of considering and evaluating alternative cell designs. The manufacturing engineering students, however, are not well-acquainted with computer simulation. They are not familiar with simulation methodologies or with the various simulation languages and software that is available.

It was proposed to me that a simulation tool be developed exclusively for use in the CIMS lab. The notion being that by limiting the scope and flexibility of a simulation system, and by making it easy to learn and understand without requiring a strong background in simulation methodologies or languages, some of the benefits of simulation could be exemplified and an appreciation of simulation developed. By devising a system where the students could make use of a simulation tool prior to actually learning about the methodologies, the terminology, the various languages and such, simulation could be incorporated into the manufacturing curricula. The students could then take courses in computer simulation if they were so inclined.

The goal of this paper then, is to explore various simulation techniques, methodologies and products, and to tailor an existing system or develop a new system specifically for simulating the FMC in the manufacturing engineering department's CIMS lab.

3.0 Simulation Software

3.1 An Overview of Simulation Software

According to Haider and Banks [1986], "Over 100 simulation software products are currently available on a wide variety of computers, from micros to mainframes." Indeed, the number of simulation software products is staggering. In this section, we'll examine and classify some of the more popular software types and identify their strengths and weaknesses.

Law and McComas [1992] identify two major classes of simulation software, *simulation languages* and *manufacturing simulators*. *A simulation language*, according to these authors, "is a computer package that is general in nature . . . but may have special features for manufacturing such as work stations and material-handling modules." Some examples of simulation languages are GPSS/H, GPSS/PC, MODSIM II, SIMAN/Cinema IV, SIMSCRIPT II.5, and SLAMSYSTEM. *A manufacturing simulator*, "in its most basic form, is a computer package that allows one to simulate a system contained in a specific class of manufacturing systems with no programming." Examples of simulators include FACTOR/AIM, Micro Saint, ProModelPC, SIMFACTORY II.S and WITNESS.

With a *simulation language*, a model is developed by writing a program using the language's modeling constructs. These constructs include entities (parts), attributes (part type or due date), resources (machines or workers) and queues (waiting areas). The model is then compiled and executed much like computer programs written in a high level programming language. The advantage of modeling with a simulation language is that practically any system can be represented, regardless of its complexity or uniqueness. The drawbacks to modeling with a simulation language are the need for programming expertise and possibly the long time spent coding and debugging.

Manufacturing simulators, on the other hand, do not require programming. The particular system of interest is selected (from the domain of the package) by choosing items from menus and filling in forms or by the use of graphics. The code is generated by the system and executed. The advantages of modeling with a manufacturing simulator are reduced development time and ease of use. The major drawback is that they are limited to modeling only those manufacturing configurations provided by the package. Thus, if a manufacturing system has some unique features or control logic, an accurate representation might not be possible. Some simulators overcome this problem by providing programming-like commands and allowing the developer to inject his own code, but then the "advantage" of this type of system is lost.

3.2 Classifications of Simulation Software

In this section we'll explore some of the variations currently encountered in today's simulation software. To do so, we'll need to become familiar with some of simulation's basic terminology. A system being modeled is said to have various states. A system's state is its components current conditions at any one time. For example, a machine might have three states (busy, idle, and broken), and a queue might have five (empty, one, two, or three waiting, and full). A system will have certain events associated with it. Events are situations that cause a change in a system's state (a machine breaks, or a part arrives). Lastly, a system will possess certain activities. Activities are processes which are performed by the system (loading a work-piece, or drilling a hole).

Haider and Banks [1986] describe three levels at which simulation software can be classified: *the system level, the application level, and the structural level.* At the system level, two types of systems are generally recognized: *continuous systems* and

discrete systems. In a continuous system, parameters (usually referred to as state variables) change continuously over time. Differential equations are used to depict the rates of change of the state variables. These equations are solved at specified time intervals to determine the current values, and the current values are assessed to determine if certain events should be scheduled. The system then processes the scheduled events and calculates the desired performance statistics.

In a *discrete system*, state variables change only at discrete points of time (usually referred to as event times). The simulation is conducted by scheduling these event times and monitoring the system states with respect to an internal clock. The scheduling and processing of events and activities are handled internally by the system when the model is run, and numerous performance measures are available. Most manufacturing systems are modeled using discrete time units and for our purpose (the CIMS lab's FMC), this approach is preferred.

At the second level of classification, *the application level*, the authors devise two categories: *special purpose software* and *general purpose software*.

This distinction was covered in the previous section. Basically, special purpose software is designed to model specific types of systems. Software products of this type are typically referred to as simulators. General purpose software allows one to model any type of system using the system's constructs. Software products of this type are generally referred to as simulation languages.

Classification of software at the *structural level* requires an understanding of how the simulation is conducted, as opposed to what can or is being modeled. At this level the modeling orientation (referred to also as the world view in some texts) that is employed by the simulation software is described. Three orientations or world views prevail: *event scheduling, process interaction, and activity scanning.*

In the *event scheduling orientation*, a system is viewed as consisting of a number of possible events (instances at which time state changes take place). Some events might be the arrival of a part, the completion of a machining task, or a part exiting a conveyor.

The modeler defines the events and develops the logic associated with each event. Events are scheduled via an internal mechanism, and processed in order. When an event is processed, its associated logic is executed, creating new events to be scheduled and changes in state variables. This is a widely used approach in the simulation of manufacturing systems (usually coupled with the discrete time, *systems level* classification).

In a *process interaction orientation*, the modeler views the system as a set of processes. The flow of a part through a manufacturing line might be a process, or the loading or unloading of conveyors. The user describes the flow of parts through these processes and the system translates this into an appropriate sequence of events. The simulation is then carried out in a manner similar to that described as event scheduling.

In an *activity scanning orientation*, the user describes the conditions necessary to start and stop each activity in the system. Time is advanced in equal increments and, at each increment, the conditions are evaluated to determine if any activities can be started or terminated. This orientation is most often associated with continuous processes where continuous measures such as temperature and pressure are monitored.

These three levels of classifications are generally not of great interest to the user of a simulation software package. For the most part, users are interested in what the software can and cannot do, what systems can be modeled and what systems cannot. The system level classification and modeling orientation is often transparent to the user - and many users, if asked, probably would not know into which classifications their simulation software falls. But it is the classification at the systems level and the orientation of the software that determines what can and cannot be modeled, and the level of expertise required. These distinctions are therefore critical in the determination of specifications for the CIMS lab's FMC simulation system.

3.3 Desirable Features of Simulation Software

In this section we'll explore what features and characteristics various researchers and authors have identified as desirable in simulation software, and identify those that are particularly applicable to the proposed CIMS lab FMC simulation system.

Law and Haider [1989] separate the desirable features of simulation software into six categories: *General features, animation, statistical capabilities, material handling modules, customer support, and output reports.* Within each category, features are listed according to the authors' assessment of their relative importance.

In the first category, general features, the authors identify the following simulation package features as desirable: *modeling flexibility, general attributes for entities, ease of model development, fast model execution speed, maximum model size, and compatibility across computer classes. Modeling flexibility* is the most important feature. "If the simulation package does not have the necessary capabilities for a particular application, then the system must be approximated, resulting in a model with unknown accuracy." It is also desirable for entities to have *general attributes* which can be appropriately changed. Not only does this contribute to flexibility in general, it allows for the study of non standard performance measures and can facilitate the modeling of more complex control strategies. *Ease of model development* is particularly important for the CIMS lab FMC project. Aside from ease of use, we also require ease of understanding. The authors prefer simulation software with *interactive debuggers* and *on-line help capabilities. Fast model execution speed, maximum model size*, and *compatibility across computer classes*, although important features in general, are not particularly critical for the CIMS lab FMC simulation tool.

"Animation has become a widely accepted part of the simulation of manufacturing systems." The authors explain its usefulness in communicating the essence of a simulation model, in debugging and verification of a model, and in their

"suggesting" of new control strategies. Desirable animation features, according to the authors, include *ease of development, user creation of high-resolution icons, and the smooth movement of icons across the screen.* I would also include the ability to vary the speed of animation, so that the dynamics of a system could be more easily understood.

Statistical capabilities are inherently important in computer simulation. Since sources of randomness are what ultimately necessitate the simulation of a manufacturing system, the simulation package being used must provide an ample assortment of statistical capabilities. A *wide variety of standard distributions* should be available, as well as the *ability to use distributions based on observed shop floor data.* Other desirable features listed under statistical capabilities include *the ability to make independent replications of the model automatically, the ability to specify a warm-up period, and the ability to construct confidence intervals* for the desired measures of performance.

Material handling systems are an important part of most manufacturing systems, and are often the focus of simulation analyses. It is therefore desirable for a simulation system to *provide flexible, easy to use modules for modeling transporters, AGVs (Automatic Guided Vehicles), conveyors, cranes and robots.* The authors note that "the existing material handling modules in some simulation packages may not always be sufficient due to the great diversity of available material handling equipment". It is therefore desirable to allow for user specified material handling components.

Customer support can be a serious consideration in selecting simulation software. Some vendors offer *general software training and/or provide technical support* for specific modeling problems encountered by their users. *Good documentation* is also desirable and should include a wide variety of detailed examples.

Finally, the authors describe the desirable features for a simulation package with respect to its output reports. A simulation package should provide the ability to quickly and easily produce *standard reports* for commonly occurring performance statistics (utilizations, queue sizes, and throughput). *Tailored reports* should also be quickly and easily produced. Indeed, the whole point of conducting a computer simulation study is to

obtain some required information. The nature of and presentation of this information should be user determined. *High-quality graphical displays* and *access to the individual model output observations* (rather than just the summary statistics) are also listed as desirable.

In addition to many of the features listed above, Haider and Banks [1986] identified the following desirable features: *input flexibility, syntax, structural modularity,* and *modeling conciseness.* The authors describe *input flexibility* as the ability "to develop models either in a batch mode or in an interactive environment . . . where the system prompts for information on pre-formatted screens." By *syntax* they refer to the software's modeling scheme. The system should be "user friendly (convenient for a user to follow and understand), consistent and unambiguous. A good syntax scheme facilitates rapid development of the model and reduces mistakes . . .". *Structural modularity* refers to the software's modeling constructs. The authors feel that simulation software products should allow modular development of a model. "Meaningful modules are system layout by segments, equipment characteristics, product flow schemes, product requirements, initial conditions, statistics collection and output report requirements. In such an environment, each component of the model can be changed without altering the others." By *modeling conciseness* they intend that the software have powerful block/node capabilities and commands which allow for the concise modeling of complex decision rules and flow patterns.

Some additional features worthy of mention are that the software "take into account issues like the speed of conveyors and automated guided vehicles . . .", Norman [1992], and that "One should be able to read and write external files directly from the simulation package without using external routines", Law and McComas [1992].

3.4 Problems Associated with Computer Simulation

> "Historically, simulation has been used primarily as a planning tool for new projects in manufacturing or major renovations. Recently, however, some organizations are applying the technology for the purpose of optimizing current operations. The whole application of simulation is going to grow and extend beyond these current uses to become integral with the daily operation of factories.
>
> Today, simulation models are developed and managed by a select group of individuals, quite often Industrial Engineers. Everyone in the organization comes to these individuals for answers from the model."
>
> - Norman [1992]

> "Have we developed languages and approaches best suited to the skilled practitioner, languages that take months or years to master?"
>
> - Smith, Cypher, and Spohrer [1994]

Aside from the aforementioned problems with simulation languages and manufacturing simulators, the problems most often cited with respect to simulation software concern the products' complexity and the level of expertise required to make use of it. As simulation software has become more versatile, it has also become more complicated and laborious to use. In this section we'll look at the problems and pitfalls associated with simulation software, and identify those problems that are most significant to the CIMS lab FMC project.

Regardless of what type of simulation software is used, a certain amount of knowledge and expertise is required. Law and Haider [1989] state that they believe "model coding will represent only 30 to 40% of the total required work in a typical sound simulation study." Other important activities they identify include project formulation, data and information collection, statistical modeling of system randomness, validation of the model, and the statistical design and analysis of the simulation runs. "Furthermore,

these tasks are, for the most part, not performed by existing simulation software, regardless of how easy these products are to use. Thus, it is incumbent on the simulation developer or user to have a fair amount of expertise in *simulation methodology* per se, in addition to the use of one or more simulation products."

In an article on software reuse, Charles Krueger [1992] introduces the concept of *cognitive distance*, and defines it as "the amount of intellectual effort that must be expended by software developers in order to take a software system from one stage of development to another." This is an important notion to keep in mind, as we desire a simulation tool requiring little background knowledge or methodological expertise.

Smith, Cypher, and Spohrer [1994] advocate systems with a property they dub *minimum translation distance*, which they describe as the conceptual distance between people's mental representations of concepts and the representations the computer will accept. They also advocate a *familiar user's conceptual model*. By this, they mean that concepts used in a system should be cast into terms the user can understand. Indeed, instead of using the standard simulation terminology such as entities, resources, attributes and queues, the engineering students might be more comfortable with terms such as parts, machines, properties and waiting areas.

Most of the forgoing was applicable to simulation languages, where the user writes programs in a particular language and must, therefore, understand the constructs and methods implored. But what of the simulators? In fact, the use of simulators still requires knowledge of simulation terminology and methodology. Models are constructed by specifying the entities, resources, processing times, etc. Simulation terminology and methodology must be understood in order to select the appropriate model components.

4.0 The CIMS lab FMC Simulation Software

4.1 Design alternatives for the CIMS lab FMC Software

Three alternative designs were conceived for the implementation of the CIMS lab FMC simulation tool. One option would be to develop a code generator - a software package that would prompt the user for the required information, and generate the code necessary to conduct the simulation in an existing simulation language. Another option would be to customize an existing manufacturing simulator - to shield the user from unnecessary complexity and make the simulator straight forward and easy to use. The last option would be to develop a simulation system that stands alone - software that prompts the user for the necessary information, carries out the simulation, and produces the results itself. In this section these three alternatives will each be discussed, developed and evaluated in turn.

The first design alternative to be evaluated is the simulation language code generator. As noted previously, simulation languages provide many features and variations of components to allow for the simulation of virtually any type of system. Programs are written in these programming languages, compiled and executed. An interface could be developed for this type of system, to shield the user from having to learn the modeling technique and syntax implored. The interface (code generator) would prompt the user for specifics about the system to be simulated (how often parts arrive, the number of machines, the processing times and queuing associated with each, etc.). The software would then produce the code required to simulate the described system in an existing simulation language. The resulting code would then have to be compiled with the language's compiler and then executed.

In considering this approach, I envisioned using the SIMAN language, with which I am familiar. One problem became immediately apparent, the conveyor scheme presently used in the CIMS lab FMC cannot be easily portrayed. While SIMAN does provide two conveyor constructs, neither is appropriate. The conveyor types provided are termed accumulating conveyors and non-accumulating (which are, incidentally, the same constructs available in most simulation languages). An accumulating conveyor is used to model systems where the conveyor keeps moving and parts accumulate at the end. Non-accumulating conveyors model systems where the distance between parts on a conveyor remains constant (the conveyor stops when a part reaches the end). The problem is that parts can only enter the conveyor at one end, and exit from the other. There is no construct available for circular type conveyors such as the CIMS lab FMC's (conveyors which allow parts to enter and exit at various points along the way). This could be modeled in SIMAN, by using a series of conveyors, one between each pair of machines, but then the situation of having "the" conveyor stop (to load, unload, or wait if specified) becomes problematic.

I decided that with SIMAN, (and most languages are similar), it is just too difficult to model the current system accurately, let alone devising schemes to allow for alterations and variations of the system. Even if a representation of the loop conveyor were contrived, requiring the user to develop a model with this software, compile and execute the resulting code using other software, and constantly have to switch back and forth between the two, seems awkward and cumbersome. For these reasons, the code generation alternative was rejected.

The second alternative, customizing an existing simulator for use in the CIMS lab, is considered next. Manufacturing simulators are supposed to provide a means of conducting simulation studies that require no programming. What I had envisioned was providing a subset of the modeling constructs available in a simulator, with default parameters preset, so that a novice user might find the system easy to understand and utilize.

To explore this alternative, I considered ProModel, a popular manufacturing simulator which is presently available on Miami's Applied Sciences network. The interested reader may wish to reference the ProModel User's Guide [1994]. One of the problems previously identified with simulators is that they can model only those systems which fit into one of the classes of systems for which the simulator provides. The CIMS lab FMC's loop conveyor again becomes a problem. The conveyor constructs available in ProModel are the same as those available in SIMAN, accumulating and non-accumulating types. Entities must enter at one end and exit at the other. As far as customizing a simulator specifically for use in the CIMS lab, ProModel seems impracticable.

Arena, another manufacturing simulator (see the Arena User's Guide, 1993), provides a feature they call templates. I thought that perhaps by constructing and utilizing templates easy simulation modeling for the CIMS lab FMC could be facilitated. With templates, one can create specific manufacturing configurations and save them. Then, if simulations of these basic configurations, with various modifications and changes are desired, they can be developed very quickly. I considered providing a number of these templates, to furnish a variety of modified CIMS lab FMC configurations from which the user could choose.

One problem with this approach is that only a limited number of templates (configurations) could be made available. Even if an assortment of templates were developed and considered sufficient, another problem is that, even with the use of templates, specific attributes and properties must be specified by the user. These specifications are made via dialog boxes that refer to everything in very jargon-intensive language (resource, entity, attribute, route, sequence, queuing capacity, etc.). The user would have to be familiar with this terminology in order to utilize the templates. The output is also very jargon-intense and cannot be reformulated. For these reasons, the 'customize an existing simulator' design alternative was also rejected.

A stand alone simulator, one that prompts the user for information and conducts the simulation itself, would need be developed.

4.2 Specifications for the CIMS Lab FMC Software

The basic purpose of the CIMS lab FMC simulation system is to provide a means for manufacturing engineering students, not familiar with simulation methodologies or terminology, to easily specify a manufacturing system cell configuration and conduct simulation studies relative to it. Thus, the desired software requires an interface which is very user-friendly and free from any unfamiliar simulation terminology. The output from the system, likewise, should be readily understandable. In this section, we'll define the specifications of the proposed software, describe what features and capabilities it will provide, and indicate the form of the interface and output reports.

At the systems level (see *classifications of simulation software*, earlier in this report) a discrete systems implementation will be used. The discrete environment works well for modeling manufacturing systems, since events of interest in manufacturing simulations (start job, finish job, load part on conveyor, etc.) generally occur at specific times. It will also be easier for the users (students) to specify service times, load/unload and transport times, in a discrete environment.

At the application level, the simulation tool would have to be classified as special purpose software. Of course, it is being designed specifically for the CIMS lab FMC. But the more significant aspect is that the user needn't write programs. There is no programming language to learn, the user simply specifies characteristics to be incorporated into the model, and the system handles the particulars.

At the structural level an event scheduling orientation, or world view, will be used. Event scheduling is a commonly used orientation for simulation of manufacturing

systems and it provides for a number of performance measures that might otherwise be difficult to derive. It may also be the easiest orientation to implement, and subsequently enhance.

Desirable features for simulation software, which were identified previously and are to be incorporated into the software, are now discussed. The desirable general features that will be provided include modeling flexibility, general attributes for entities, and ease of model development. Both aggregate level and detailed level simulations can be carried out, and modeling flexibility will be provided at each level. At the aggregate level, the user can specify the number of pallets, the number of machines, and the number of various types of machines. At the detailed level, the user specifies the processing time distributions for each machine and robot, and the type of *station-conveyor interaction mode* (to be discussed later) for each machine. The general attributes for entities feature will be useful to me as I implement the system, but will be transparent to the user (since they won't be programming). Ease of model development, as previously mentioned, is of primary importance to the CIMS lab FMC project and will of course be accommodated. Friendly, easy to use and understand interfaces and reports will also be provided.

The remaining desirable general features, fast execution speed, maximum model size, and compatibility across computer classes, are not considered to be of vital importance to the CIMS lab FMC project. Animation capabilities will not be provided.

Statistical capabilities provided for include a wide variety of standard distributions, and the ability to perform independent replications automatically. Constant, uniform, normal, exponential, triangular, and user defined distributions will be available for specifying robot loading and unloading, and machine service times. The user can specify how many replications to be conducted, and if a number of replications are conducted, summary statistics and confidence intervals will be automatically computed. The length of each replication will determined by the user, as well as the length of the warm-up period. The warm-up period will apply to only the first replication, subsequent replications will start in the state in which the previous replication

ended. Output statistics will include, for each replication, the number of parts completed and the mean, standard deviation, minimum and maximum values for time-in-system and time-on-conveyor. Utilization Statistics will be computed for each of the stations, and the conveyor usage will be measured in terms of the average number of parts on the conveyor throughout each replication. Summary statistics for multiple replications will include mean, standard deviation, and 95% confidence intervals for number of parts completed, time-in-system, and time-on-conveyor, as well as average utilizations of stations and average number of parts on the conveyor throughout all of the replications.

The desirable feature of having a wide variety of material handling modules provided is of primary importance to most simulation practitioners, since material handling systems can be very difficult to represent. In our case, the primary material handler, the conveyor, will be alterable in three ways. One option is to have the conveyor stop and wait while parts are serviced, as is currently the case. This option will be referred to as the *conveyor waits* station/conveyor interaction mode. Another option is to have the conveyor to stop only while a part is being loaded or unloaded by a robot, then continue along (transporting an empty pallet, as well as any other pallets), while service takes place. This practice will be referred to as the *conveyor continues* station/conveyor interaction mode. Lastly, the user might specify pallet waiting areas, where a pallet would reside while the robot and servicing of parts takes place. This last station/pallet interaction mode, dubbed *pallet waits,* requires an additional piece of machinery, but minimizes the amount of time that the conveyor is stopped. In this type of FMC, minimizing the conveyor's stop time is very much desirable.

4.3 Description of the CIMS Lab's FMC Simulation Software

Particular characteristics of the CIMS lab FMC Simulation Software are now discussed. Manufacturing configurations which can be simulated using the CIMS lab FMC Simulation Software will somewhat resemble the FMC's current configuration. Parts will arrive via the automated storage and retrieval system, and travel along a circular conveyor on pallets. When a part arrives at a station which it needs and is available, the conveyor stops and the part is unloaded by a robot from the pallet and onto the machine for service (according to the station's conveyor interaction mode). As service is completed, the part is loaded back onto the pallet by a robot (again, according to the station's conveyor interaction mode), and travels along the conveyor to the other stations.

The length of the conveyor , the speed of the conveyor, and the number of machines along the conveyor are all specified by the user, but the spacing of machines along the conveyor will be determined by the system (equidistant). Since the loading/unloading robot and automated storage and retrieval system pose a potential bottleneck, the user can specify that completed parts be removed from the system via the same robot that loads unfinished parts (as is the current practice), or that a separate removal station and robot be included.

The software utilizes three input screens whereby the user specifies the desired systems' configuration, the processing requirements of parts, and the simulation parameters and output designations. Each of these screens are depicted here, followed by descriptions of the input information and verification and limits thereof.

screen 1:

```
                    MAIN MENU FOR SYSTEM CONFIGURATION

Station   Station   Conveyor    Robot time          Service time
number    name      /Pallet     distribution        distribution
                    behavior    (in seconds)        (in seconds)
-------   -------   --------    ------------        ------------
<0>       enter     W/W         Norm(36,12)
<1>       lathe     C/C         Norm(36,12)         Tria(60,90,100)
<2>       drill     C/W         Cons(45)            Expo(200)
<3>       remove    (the load station removes the finished parts)

<4> Conveyor Length:  60 feet.
<5> Conveyor Speed:   90 seconds to complete one revolution.
<6> Pallet load/unload time (C/W only):  10 seconds.

<7> add a machine
<8> remove a machine

<9> load a configuration (previously saved) from file
<10> next screen
<11> quit

enter a line number to edit that line's information
  or to invoke the desired option:
```

The first screen allows the user to specify the FMC configuration to be simulated. In this example there are 3 stations along the conveyor: the enter station, the lathe station, and the drill station. The enter station is always present. The system can accommodate up to 9 work stations, as well as a separate removal station. In this example a separate removal station is not specified. This is shown by the statement that the load station removes the finished parts. If a separate unloader were specified, it's Conveyor/Pallet behavior and Robot time distribution would be displayed. For each station, the station's name, Conveyor/Pallet behavior, Robot time and service time distributions are displayed.

To edit any of the displayed information, or to invoke a desired option (listed on the lower portion of the screen), the user enters the integer line number (in norkey brackets, <>) preceding the information or option. For instance, the user could edit the lathe station information by entering 1, change the conveyor speed by entering 5, or quit by entering 11.

A station's Conveyor/Pallet behavior is specified as W/W, C/C, or C/W. These correspond to the Station/Conveyor interaction modes previously mentioned. These abbreviated designations were developed to save screen space and describe more precisely how the station and conveyor interact (in terms of what waits). Specifically, W/W signifies that the conveyor and pallet both wait. Formerly referred to as the *conveyor waits* station/conveyor interaction mode, this means that when a part arrives at a station for service, the conveyor stops and remains stopped while the part is loaded, serviced, and unloaded. The conveyor and pallet both wait for the part. C/C signifies that the conveyor and pallet both continue. This interaction mode, dubbed *conveyor continues* in the preceding discussion, specifies that when a part arrives at a station for service, the conveyor stops while the part is loaded, then the conveyor and empty pallet continue along while service takes place. When service is completed, the part awaits an empty pallet. When an empty pallet arrives, the conveyor stops and waits while loading occurs. The conveyor and pallet are then released. C/W signifies that the conveyor continues, the pallet waits. Previously denoted as the *pallet waits* station/conveyor interaction mode, here, when a part arrives at a station for service, the conveyor stops. The part and pallet are pushed onto a pallet waiting area, and the conveyor is released. This operation takes 10 seconds, specified in line <6> of the input screen, Pallet load/unload time. The pallet remains in the loading area and the conveyor continues along during the loading, service, and unloading times. When the part has been reloaded onto the pallet, the conveyor is again stopped for 10 seconds, the part and pallet are pushed back onto the conveyor, and the conveyor is released.

The robot and service time distributions available (from a separate input screen) include constant, normal, uniform, exponential, triangular, and user-defined distributions. The user selects a distribution and enters the appropriate parameters. Integer values are used (since the simulation is conducted in seconds it was decided that 'whole seconds' provided enough precision) and various edit checks are performed to ensure that the parameters make sense. For instance, if a triangular distribution is selected, there are 3 parameters which must be positive integers and conform to P1 <= P2 <= P3.

Conveyor length, conveyor speed, and pallet load/unload time are all user specified, the only requirement being that they be positive integers.

Options 7 & 8, add a machine and remove a machine were included to allow the user to quickly and easily alter the current configuration. The add a machine option allows the user to duplicated an existing machine, and place it anywhere along the conveyor. This works out well if one is trying to minimize bottlenecking or adding machines similar to those present. Option 9, load a configuration (previously saved) from file, allows the user to quickly modify all of the displayed information, provided the desired configuration has been previously saved. Configurations are easily saved via another screen.

When machines are added or removed, or conveyor length or speed is altered, the system automatically recalculates the time required to move from one station to the next. This calculated value is used by the system to schedule part arrivals, and is not accessible or of concern to the user.

screen #2:

```
                    MAIN MENU FOR SIMULATION

<1>  Maximum Number of Pallets Allowed:        4

<2>  Processing by machines is to be done    IN THIS ORDER
<3>  Machining operations required:              enter
                                                 lathe
                                                 drill
                                                 remove
<4>  Number of Replications:        5
<5>  Length of each replication:    7200 seconds      (2 hours)
<6>  Warm-up period:                3600 seconds      (1 hour)

<7>  Previous screen
<8>  next screen

enter a line number to edit that line's information
  or to invoke the desired option:
```

The second screen allows the user to specify the number of pallets to be allowed in the system, the part's machining requirements, and some simulation parameters. It functions as the first screen does (indeed, as all screens do), the user entering the line number to edit some particular information, or to invoke a desired option. The first line, <1> Maximum number of pallets allowed, allows the user to specify, obviously, the maximum number of pallets, which must be between 1 and 20. The maximum of 20 was established because of the edit checking required and my belief that with more than 20 pallets, the loop conveyor and queueless station configuration doesn't really make sense.

Line <2> Processing by machines is to be done, in this example is set to IN THIS ORDER, as opposed to IN ANY ORDER. In THIS ORDER mode, parts are required to obtain the services of the stations listed in <3> specifically in the order listed. In ANY ORDER mode, a part can obtain service from any available station in any order

that it finds them available. The system automatically checks to insure that the operations listed in <3> machining operations required match stations defined on the previous screen. Requiring an operation for which no station exists would result in 0 parts completed (the system frozen in a state of having whatever maximum number of parts, not needing any of the idle machines, circling around and around...). If operations are specified for which no machine exists, the user is alerted to this fact and forced to edit the machining operations required.

The number of replications is user specified, an integer between 1 to 20. It was thought that 20 replications would be sufficient to measure the performance of a given configuration.

The length of each replication, and of the warm-up period are shown both in seconds and hours. When opting to change either replication length or warm-up length, the user enters the duration in hours. The actual value utilized by the system will be the in-seconds equivalent, but it seemed adequate for the user to specify in hours. A 10 hour limit is imposed because of C++'s integer representation size limit.

screen #3:

```
|                          MAIN MENU FOR OUTPUT SPECIFICATION                  |
|                                                                             |
|<1>  Output file:     not specified                                          |
|                      Output will be displayed on screen, but NOT saved      |
|<2>  Debugger:        OFF.                                                   |
|                                                                             |
|<3>  Replications:    5                                                      |
|                                                                             |
|<4>  Previous screen                                                         |
|<5>  Run the simulation                                                      |
|                                                                             |
|<6>  Save the current configuration to file                                  |
|<7>  Quit, DO NOT RUN the current configuration                              |
|                                                                             |
|enter a line number to edit that line's information                          |
|  or to invoke the desired option:                                           |
|_____________________________________________________________________________|
```

This last input screen allows the user to specify an output file, save the current configuration, and/or toggle the debug feature. In this example, no output file is specified, so that output will be displayed on the screen but not saved. To save the output to file, the user would edit line 1. The system would then prompt the user for a file name (or DOS path), and create and/or open the file.

The debugging feature produces a listing of the system's calendars at each time increment. It was useful to the developer, but might not be understandable to the average user. This feature was left in for the use of future developers, and an understanding of simulation techniques (particularly event scheduling) would be required for meaningful interpretation. Debug output is sent to a file named sim0.dbg.

Given the facts that the debugger produces a large volume of output, and that an unaccustomed user might want to experiment with it, the system checks to see how many replications are planned when the debugger is turned on. If more than 1 replication is planned the user is warned that much output is produced in debug mode and that perhaps only 1 replication should be run. The option of changing the number of replications is provided again on this screen for convenience.

Option 6 allows the user to save the current configuration to file, so that the current information can be quickly and easily re-used at a later time. If selected, the user is prompted for a file name (or DOS path) and all of the information from all 3 input screens is saved to the specified file. This can be done prior to or following a run. The system echoes back the file name as verification that the configuration was saved.

The user is allowed to switch back and forth between the input screens as often as is desired. Eventually, though, a simulation run is expected. The output, be it displayed on the screen or saved to a file, is fundamentally the same. The next few pages describe the output screens, and the information they present.

replication output screen:

```
|Results of replication #3:                                                    |
|                                                                              |
|length of simulation:                    3600                                 |
|number of parts completed:               32                                   |
|                                                                              |
|                          min      max      avg      S.D                      |
|                                                                              |
|time parts spent in system:  1203     1634     1407.2   112.54                |
|time parts spent on conveyor:  260     810      439.5    108.6                 |
|                                                                              |
|average number of parts of conveyor:              1.8                        |
|Utilizations of stations:                                                     |
|                                  enter:   0.31                               |
|                                  lathe    0.83                               |
|                                  drill    0.54                               |
|                                                                              |
|press enter for next screen:                                                  |
|______________________________________________________________________________|
```

The output screen displays the length of the simulation and number of parts completed. Of all the parts that were completed during a particular replication, the amount of time that each spent in the system, and on the conveyor is observed. Output statistics are calculated for these observations and the minimum, maximum, average value and standard deviation of each are displayed. We see from this example, that 32 parts were completed. These parts averaged 1407.2 seconds in the system, of which an average of 439.5 seconds was spent on the conveyor.

The average number of parts on the conveyor is also computed, in this case throughout the replication an average of 1.8 parts were traveling along the conveyor at any time. Utilizations of the various stations are calculated as well. In this example, the lathe station is being utilized 83% of the time. This might explain why parts tended to spend 439.5 seconds on the conveyor, and why the conveyor averaged 1.8 parts. By adding another lathe, or shortening it's service time or otherwise decreasing it's

utilization, throughput could be increased and the time parts spend in the system and on the conveyor might be reduced.

overall output screen:

Overall Results:		
Number of Replications:		10
Length of each replication:		3600
Average number of parts completed:		37.10
Standard Deviation:		0.99
95% C.I. for mean parts completed:		(36.72,37.48)
Average Time Parts spent in system:		1555.99
Standard Deviation:		18.99
95% C.I. for mean parts completed:		(1549.00,1562.99)
Average Time parts spent on conveyor:		37.10
Standard Deviation:		0.99
95% C.I. for mean parts completed:		(36.72,37.48)
Average number of parts on conveyor:		1.17
Average utilization of stations:	enter:	0.31
	lathe:	0.82
	drill	0.54

When multiple replications are run, the system computes averages for number of parts completed, time in system, time on conveyor, average number of parts on conveyor, and station utilizations. Standard deviations and 95% confidence intervals are computed for number of parts completed, time in system, and time on conveyor.

4.4 Verification

Verification of the CIMS lab's FMC Simulation Software was accomplished by devising test case scenarios and modeling them in both the Siman simulation language and with the CIMS FMC Software. A total of six test cases were developed. For each of the station/conveyor interaction modes provided by the software, two test case simulation studies were conducted. The first test case was conducted with constant robot and service times, the second with random robot and service times.

While Siman provides for much flexibility and allows for the representation of many manufacturing environments, an accurate modeling of the CIMS lab's FMC is difficult. To portray the conveyor, which has to service several queueless stations and be able to be stopped by any entity at any time, a series of station blocks were used. The station blocks used to represent the conveyor are located between each of the actual stations, and allow for the delays that one entity causes another in this type of conveyor layout. The CIMS FMC software handles things a little differently, with a separate events calendar expressly used for scheduling parts along the conveyor and delaying them appropriately.

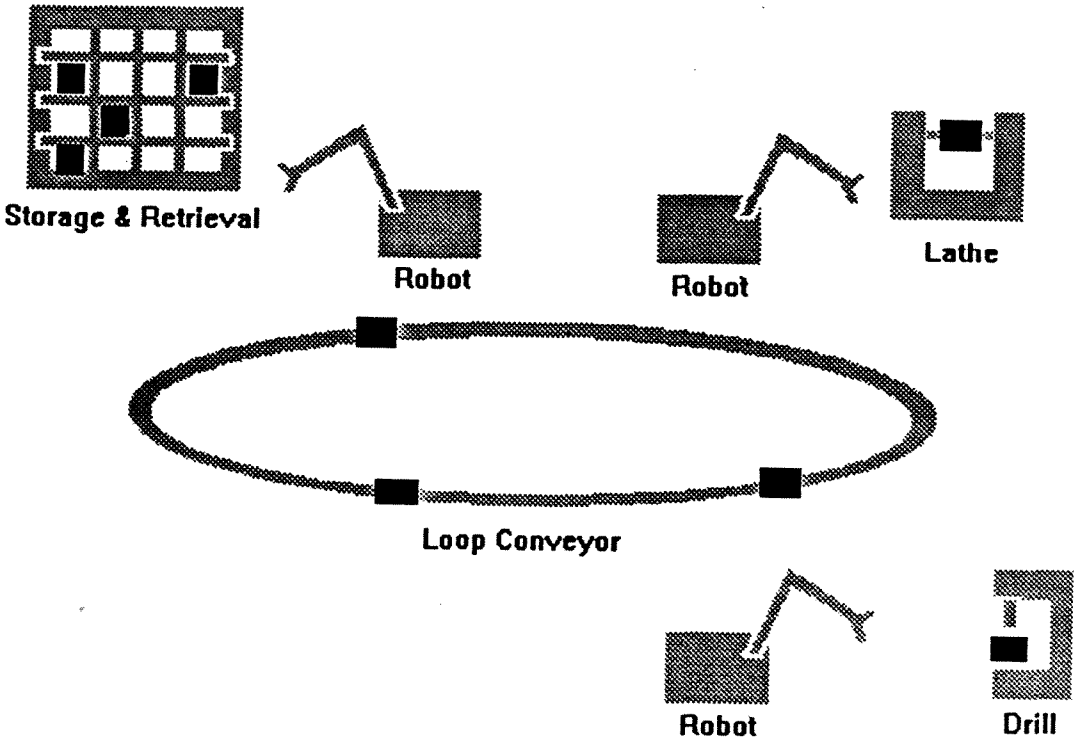
These different approaches of modeling the conveyor produce slightly different timings and orderings of events. The discrepancy is minor and is not evident in the test case scenarios where the conveyor is often in a stopped state. Indeed, most of the test case scenarios show very similar results. With the *conveyor continues* station/conveyor interaction mode however, the timing and ordering variations result in noticeably different statistics. This is the station/conveyor interaction where parts are loaded from pallet to station by robot and empty pallets circulate around the conveyor. These empty pallets are delayed just as full pallets are, but now the delaying, timing, and ordering of events is twice as noticeable. In this station/conveyor interaction mode, a finished part has had to traverse the conveyor on a pallet (as it would in any interaction mode) and it has also had to wait at each station for an empty pallet to arrive. Despite this

discrepancy, the CIMS FMC software and Siman test cases compare well, producing nearly identical results in the *conveyor waits,* and *pallet waits* models, and very similar results in the *conveyor continues* models.

For each test case, a description of the configuration, a table comparing the CIMS FMC software and Siman results, and a statistical comparison of the two simulation studies' mean throughputs has been completed. Siman models of the test cases with randomness are found in appendix A. The CIMS lab's FMC software in appendix B.

TEST CASE 1

The first test case consists of a loading station, a lathe, and a drill. The station/conveyor behavior is that the conveyor waits. All robot processing times are constant, 42 seconds, as are the service times for the lathe and drill, 207 and 526 seconds, respectively. Conveyor speed is set to 90 seconds. In this test case, the CIMS software and the Siman model produced identical results, which are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		22.5	22.5
Average amount of Time in System:		1920.0	1920.0
Average amount of Time on Conveyor:		935.0	935.0
Average number of Parts on Conveyor:		1.46	1.46
Average Station Utilization:	enter:	0.13	0.13
	lathe:	0.45	0.45
	drill:	0.95	0.95

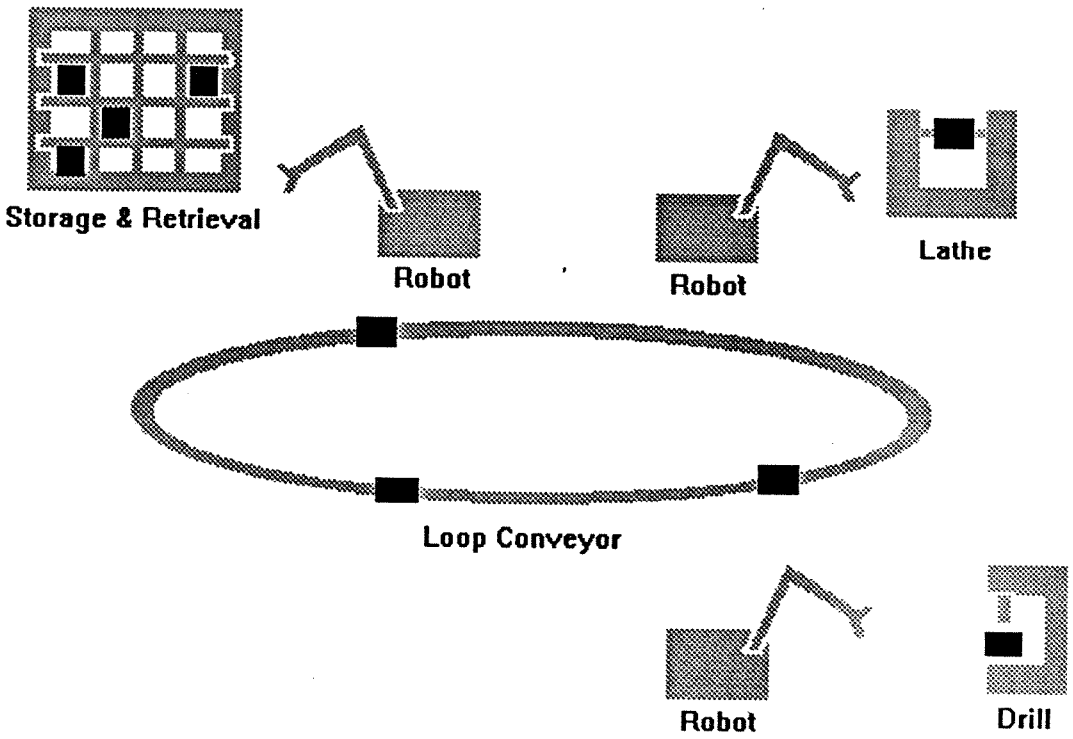
Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	23,22,23,22,23, 22,23,22,23,22	23,22,23,22,23, 22,23,22,23,22
Y-bar:	22.5	22.5
S-squared:	.2778	.2778
Ho:	CIMS mean = Siman mean	
Ha:	CIMS mean <> Siman mean	
Sp-squared:	.2778	
t-star:	0	
T.S.:	2.101	

Conclusion: fail to reject Ho, concluding with 95% confidence that CIMS mean throughput is not statistically different than SIMAN mean throughput.

TEST CASE 2

The second test case consists of the same stations as test case 1, a loading station, a lathe, and a drill. The station/conveyor behavior is that the conveyor waits (W/W). All robot processing times are governed by the uniform distribution (with parameters 40, 60). Service times are governed by the normal(200,10) and triangular(500,525,600) distributions, for the lathe and drill respectively. Conveyor speed is set to 90 seconds. In this test case, the CIMS software and the Siman model produced nearly identical results, which are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		21.5	21.4
Average amount of Time in System:		2006.8	2016.6
Average amount of Time on Conveyor:		967.8	976.6
Average number of Parts on Conveyor:		1.45	1.45
Average Station Utilization:	enter:	0.15	0.15
	lathe:	0.45	0.44
	drill:	0.96	0.95

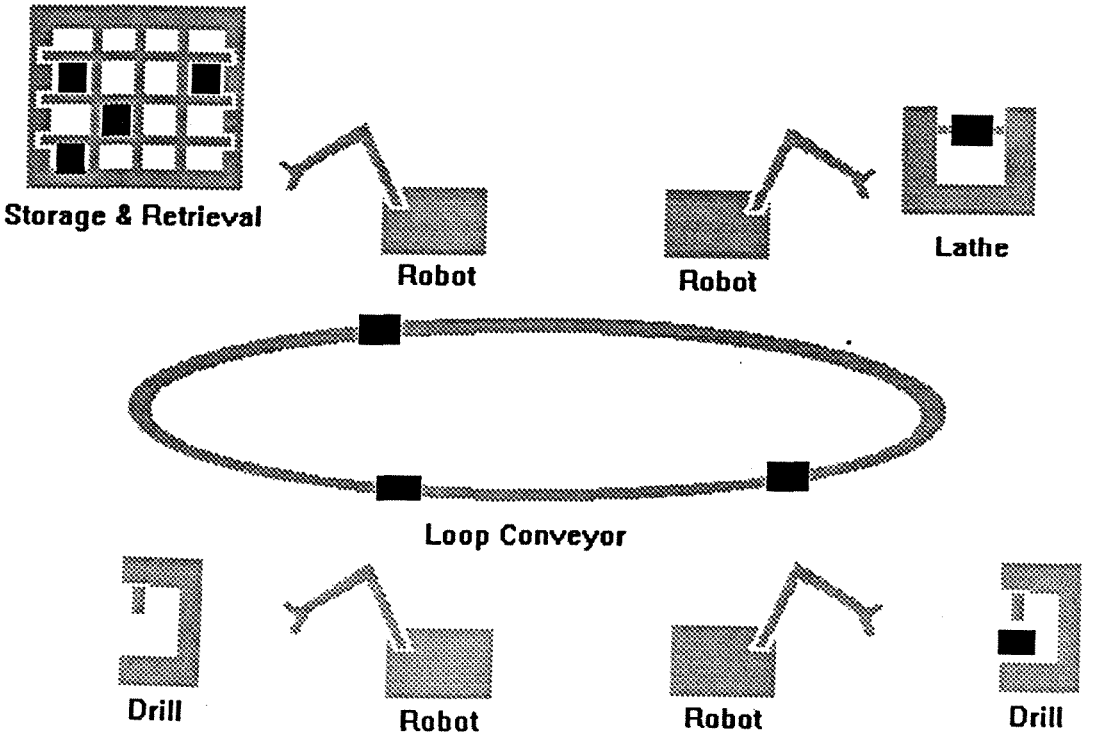
Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	21,22,21,22,21, 22,21,22,21,22	21,21,22,21,22, 21,21,22,21,22
Y-bar:	21.5	21.4
S-squared:	.2778	.2667
Ho:	CIMS mean = Siman mean	
Ha:	CIMS mean <> Siman mean	
Sp-squared:	.2723	
t-star:	.8213	
T.S.:	2.101	

Conclusion: fail to reject Ho, concluding with 95% confidence that the mean throughput is not statistically different in the CIMS Software and the SIMAN results.

TEST CASE 3

The third test case consists of a loading station, a lathe, and two drills. The station/conveyor behavior is that the conveyor continues (C/C). With the conveyor continuing along while service takes place, duplicating stations is beneficial. All robot processing times are constant (42 seconds), as are the service times for the lathe and drill, 207 and 526 seconds, respectively. Conveyor speed is set to 120 seconds. In this test case, the CIMS software and the Siman model produced nearly identical results, which are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		31.5	31.5
Average amount of Time in System:		1830.2	1841.0
Average amount of Time on Conveyor:		703.9	720.9
Average number of Parts on Conveyor:		1.54	1.56
Average Station Utilization:	enter:	0.18	0.18
	lathe:	0.81	0.75
	drill:	0.77	0.75
	drill:	0.69	0.75

Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	33,31,32,32,30, 32,31,31,32,31	33,33,31,30,31, 32,30,31,32,32
Y-bar:	31.5	31.5
S-squared:	.7222	1.1667

Ho: CIMS mean = Siman mean

Ha: CIMS mean <> Siman mean

Sp-squared: .9445

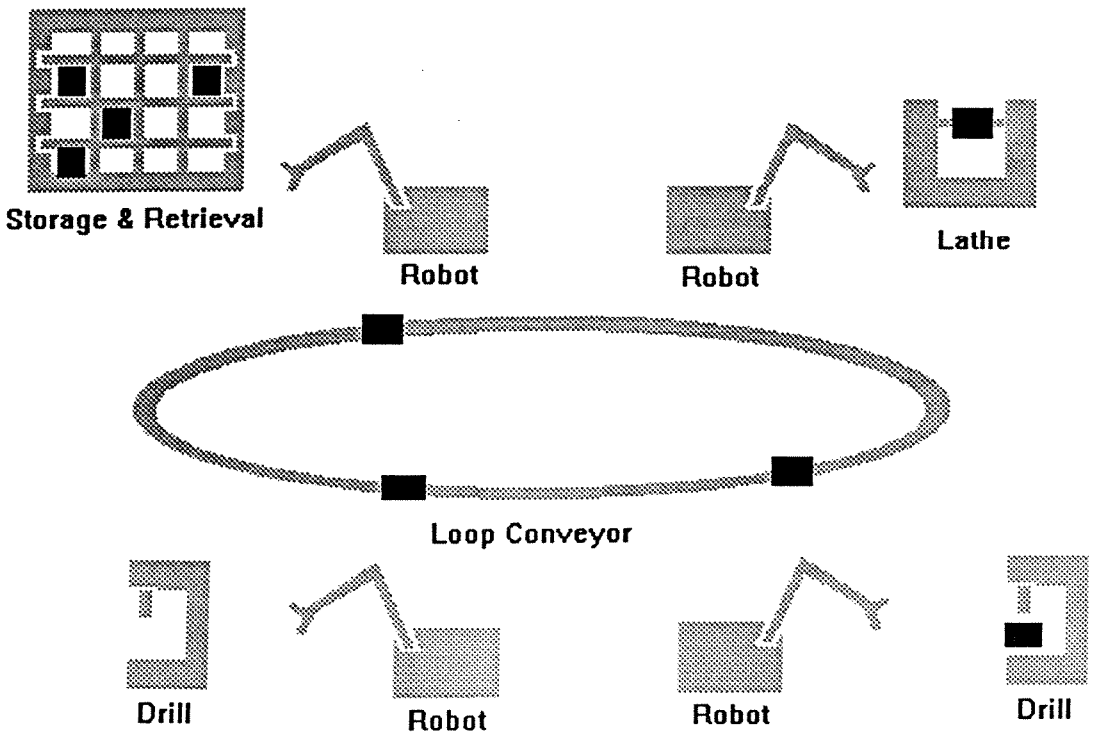
t-star: 0

T.S.: 2.101

Conclusion: fail to reject Ho, concluding with 95% confidence that mean throughput is not statistically different in the CIMS Software and the SIMAN results.

TEST CASE 4

The fourth test case consists of the same stations as test case 3, a loading station, a lathe, and two drills. The station/conveyor behavior is the conveyor continues (C/C). All robot processing times are governed by the uniform distribution (with parameters 40, 60). Service times are governed by the normal(200,10) and triangular(500,525,600) distributions, for the lathe and drill respectively. Conveyor speed remains 120 seconds. In this test case, the CIMS software and the Siman model produced somewhat different results. The discrepancy is due differences in the way the conveyor is represented in the two models, and variations in the way scheduling is done. The Siman model does not insure that 'leave service' and 'finish service' events are processed before 'arrival' events are, and tends to keep parts on the conveyor longer. The results are still fairly similar, and are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		31.4	29.7
Average amount of Time in System:		1837.4	1948.4
Average amount of Time on Conveyor:		640.4	756.6
Average number of Parts on Conveyor:		1.39	1.54
Average Station Utilization:	enter:	0.22	0.20
	lathe:	0.82	0.76
	drill:	0.81	0.75
	drill:	0.76	0.73

Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	30,34,32,31,30, 31,32,32,30,32	31,29,29,30,30 30,29,30,28,31
Y-bar:	31.4	29.7
S-squared:	1.6	0.9

Ho: CIMS mean = Siman mean

Ha: CIMS mean <> Siman mean

Sp-squared: 1.25

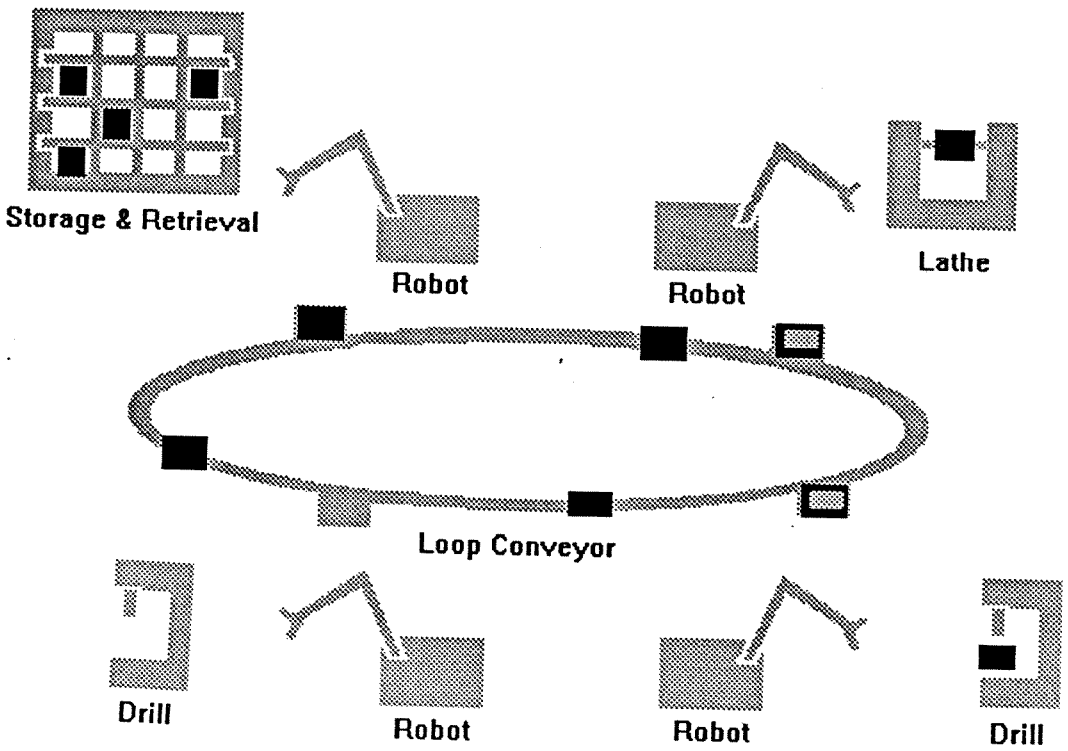
t-star: 3.0411

T.S.: 2.101

conclusion: reject Ho, concluding with 95% confidence that mean throughput is statistically different in CIMS Software and SIMAN results.

TEST CASE 5

The fifth test case consists of a loading station, a lathe, and two drills. The station/conveyor behavior is that the pallet waits (C/W). All robot processing times are constant (42 seconds), as are the service times for the lathe and drill, 207 and 526 seconds, respectively. Conveyor speed is set to 120 seconds. In this test case, the CIMS software and the Siman model produced nearly identical results, which are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		40.0	40.0
Average amount of Time in System:		1444.5	1444.1
Average amount of Time on Conveyor:		399.5	399.1
Average number of Parts on Conveyor:		1.11	1.10
Average Station Utilization:	enter:	0.29	0.29
	lathe:	0.86	0.86
	drill:	0.87	0.87
	drill:	0.87	0.87

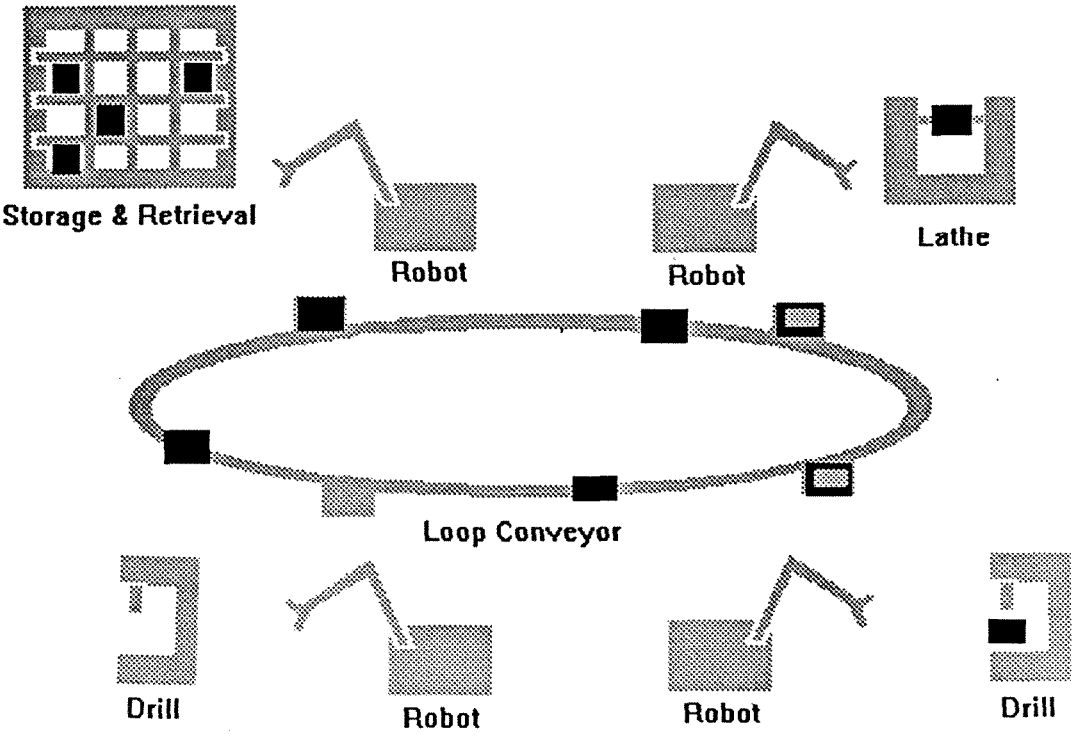
Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	40,40,40,40,40, 40,40,40,40,40	40,40,40,40,40, 40,40,40,40,40
Y-bar:	40	40
S-squared:	0	0
Ho:	CIMS mean = Siman mean	
Ha:	CIMS mean <> Siman mean	
Sp-squared:	0	
t-star:	0	
T.S.:	2.101	

conclusion: fail to reject Ho, concluding with 95% confidence that mean throughput is not statistically different in CIMS Software and SIMAN results.

TEST CASE 6

The sixth test case consists of the same stations as test case 5, a loading station, a lathe, and two drills. The station/conveyor behavior remains pallet waits (C/W). All robot processing times are governed by the uniform distribution (with parameters 40, 60). Service times are governed by the normal(200,10) and triangular(500,525,600) distributions, for the lathe and drills respectively. Conveyor speed is set to 120 seconds. . In this test case, the CIMS software and the Siman model produced nearly identical results, which are summarized on the next page.



Comparison of results:

		CIMS Software	Siman
Average number of Parts Completed:		37.1	37.5
Average amount of Time in System:		1555.9	1538.2
Average amount of Time on Conveyor:		455.8	439.7
Average number of Parts on Conveyor:		1.17	1.16
Average Station Utilization:	enter:	0.31	0.31
	lathe:	0.82	0.83
	drill:	0.85	0.85
	drill:	0.84	0.85

Statistical Comparison of mean throughput:

	CIMS Software	Siman
observations:	36,38,36,36,38, 38,36,38,37,38	36,40,36,38,38, 36,38,38,38,37
Y-bar:	37.1	37.5
S-squared:	0.9889	1.6111

Ho: CIMS mean = Siman mean

Ha: CIMS mean <> Siman mean

Sp-squared: 1.3

t-star: -0.6880

T.S.: 2.101

conclusion: fail to reject Ho, concluding with 95% confidence that mean throughput is not statistically different in CIMS Software and SIMAN results.

4.5 Conclusion and Future Works

The CIMS lab's FMC Simulation Software models manufacturing systems resembling the one in the manufacturing engineering department's Computer Integrated Manufacturing Systems lab well. It adds a number of station/conveyor interaction schemes not currently available on the actual system, and shows that throughput is very much affected by the station/conveyor interaction used. It allows the user to add machines and simulate manufacturing processes that are much more complicated than the actual manufacturing system cell could accommodate.

The software is limited in that it can model only manufacturing cell configurations closely resembling the CIMS lab's FMC. The modeling of such cell configurations is easily accomplished with the CIMS FMC software, but what of other cell configurations? What results could be obtained by providing queuing at various stations? Perhaps the conveyor needn't ever be stopped, pallets might be routed off of the conveyor at station sites without requiring the conveyor to stop.

The software could be expanded to allow for numerous part types. The basic configuration might be made more sophisticated, with sub-conveyors or shared robots. What if the whole loop conveyor arrangement was abandoned?

There are several ways in which the software could be enhanced. With increased flexibility, however, comes increased complexity. The original premise was, after all, to provide a tool for the novice user. A software system that would allow someone with little or no knowledge of simulation techniques or terminology to experiment with cell configurations, design considerations and processing requirements. Toward this end, perhaps the most beneficial enhancement would be a graphical user interface and/or animation capabilities.

5.0 Bibliography

Cheng, T. C. E. "Simulation of Flexible Manufacturing Systems." Simulation (December 1985): 299-303.

Haider, S. Wali, and Banks, Jerry. "Simulation Software Products For Analyzing Manufacturing Systems." Industrial Engineering 18 (July 1986): 98-103.

Hoover, Stewart V., and Perry, Ronald F. Simulation, A Problem-Solving Approach. Addison-Wesley Publishing Company, Inc., 1990

Krueger, Charles W. "Software Reuse." ACM Computing Surveys 24-2 (June 1992): 131-179.

Law, Averill M., and Haider, S. Wali. "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey." Industrial Engineering 28 (May 1989): 33-36.

Law, Averill M., and McComas, Michael G. "Pitfalls To Avoid In The Simulation Of Manufacturing Systems." Industrial Engineering 21 (May 1989): 28-31+.

Law, Averill M., and McComas, Michael G. "How To Select Simulation Software For Manufacturing Applications." Industrial Engineering 24 (July 1992): 29-35.

Mott, Jack, and Tumay, Ken "Developing A Strategy For Justifying Simulation." Industrial Engineering 28 (July 1992): 38-42.

Musil, David C., and Akbay, Kunter S. "Improve Efficiency Of A FMS Cell Through Use Of A Computer Simulation Model." Industrial Engineering 21 (November 1989): 28-34.

Norman, Van B. "Future Directions In Manufacturing Simulation" Industrial Engineering 24 (July 1992): 36-37.

Nwoke, Ben U., and Nelson, Ben R. "An Overview of Computer Simulation in Manufacturing." Industrial Engineering 25 (July 1993): 43-45.

Smith, David Canfield; Cypher, Allen; and Spohrer, Jim. "KIDSIM: Programming Agents Without a Programming Language." Communications of the ACM 37-7 (July 1994): 55-67.

Winton, Wayne L. Operations Research, Application & Algorithms, PWS-KENT Publishing Company, 1991.

6.1 Appendix A

Siman Test Case Code

```
          BEGIN;
          CREATE;

          ASSIGN:STA0BUS=0:
                 STA1BUS=0:
                 STA2BUS=0:
                 PIS=1:
                 MPIS=3:
                 INTERSTA=30:
                 DT=1:
                 STOPREQ=0:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME):
                 NEXT(LSTA0);

ENTRY     ASSIGN:PIS=PIS+1:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME);

LSTA0     QUEUE,STA0Q;
          SEIZE:STA0;
          ASSIGN:STA0BUS=1;
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA0A,YES;


LSTA0A    BRANCH,1:
            IF,((NEED1==1).AND.(NEED2==1)),LSTA0B,YES:
            ELSE,DONEPART,YES;

DONEPART  ASSIGN:TCT=TCT+(TNOW-ARCONV);
          DELAY:UNIF(40,60);
          COUNT:PARTS,1;
          TALLY:1,INT(ARTIME);
          TALLY:2,TCT;
          ASSIGN:NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME);

LSTA0B    DELAY:UNIF(40,60);
          RELEASE:STA0;
          ASSIGN:STA0BUS=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC1,YES:
            ALWAYS,GOCON,NO;

LSTA1     BRANCH,2:
            IF,PIS<MPIS,ENTRY,NO:
            ALWAYS,LSTA1A,YES;

LSTA1A    BRANCH,1:
            IF,STA1BUS==1,LC2,YES:
            ELSE,LSTA1B,YES;
```

```
LSTA1B    QUEUE,STA1Q;
          SEIZE:STA1;
          ASSIGN:STA1BUS=1:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED1=0;
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA1C,YES;

LSTA1C    DELAY:UNIF(40,60);
          DELAY:NORM(200,10);
          DELAY:UNIF(40,60);
          RELEASE:STA1;
          ASSIGN:STA1BUS=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC2,YES:
            ALWAYS,GOCON,NO;

LSTA2     BRANCH,1:
            IF,STA2BUS==1,LC0,YES:
            ELSE,LSTA2A,YES;

LSTA2A    QUEUE,STA2Q;
          SEIZE:STA2;
          ASSIGN:STA2BUS=1:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED2=0;
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA2B,YES;

LSTA2B    DELAY:UNIF(40,60);
          DELAY:TRIA(500,525,600);
          DELAY:UNIF(40,60);
          RELEASE:STA2;
          ASSIGN:STA2BUS=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC0,YES:
            ALWAYS,GOCON,NO;

LC1       ASSIGN:CON1T=0;
R1        QUEUE,C1Q;
          SEIZE:C1;
          DELAY:DT;
          ASSIGN:CON1T=CON1T+1;
          RELEASE:C1;
          BRANCH,1:
            IF,CON1T==INTERSTA,LSTA1,YES:
            ELSE,R1,YES;

LC2       ASSIGN:CON2T=0;
R2        QUEUE,C2Q;
          SEIZE:C2;
          DELAY:DT;
          ASSIGN:CON2T=CON2T+1;
          RELEASE:C2;
          BRANCH,1:
```

```
            IF,CON2T==INTERSTA,LSTA2,YES:
            ELSE,R2,YES;

LC0         ASSIGN:CON0T=0;
R0          QUEUE,C0Q;
            SEIZE:C0;
            DELAY:DT;
            ASSIGN:CON0T=CON0T+1;
            RELEASE:C0;
            BRANCH,1:
              IF,CON0T==INTERSTA,LSTA0,YES:
              ELSE,R0,YES;

GOCON       BRANCH,1:
              IF,STOPREQ==1,ENABLE,YES:
              ELSE,DONT,YES;

ENABLE      ALTER:C1,MPIS:
                  C2,MPIS:
                  C0,MPIS;

DONT        ASSIGN:STOPREQ=STOPREQ-1:
                   DISPOSE;

STOPCON     BRANCH,1:
              IF,  STOPREQ==0,DISABLE,YES:
              ELSE,DONOT,YES;

DISABLE     ALTER:C1,-MPIS:
                  C2,-MPIS:
                  C0,-MPIS;

DONOT       ASSIGN:STOPREQ=STOPREQ+1:
                   DISPOSE;

DELENT      DELAY:DT:NEXT(LSTA0);
```

```
BEGIN;
DISCRETE,,8,6;
PROJECT,TC102,Don Anderson;


ATTRIBUTES:ARCONV:
           ARTIME:
           NEED1:
           NEED2:
           CON1T:
           CON2T:
           CON0T:
           TCT;

RESOURCES: STA0:
           C1,3:
           STA1:
           C2,3:
           STA2:
           C0,3;

QUEUES:    STA0Q:
           C1Q:
           STA1Q:
           C2Q:
           STA2Q:
           C0Q;

VARIABLES: STA0BUS:
           STA1BUS:
           STA2BUS:
           DT:
           INTERSTA:
           STOPREQ:
           PIS:
           MPIS;

PARAMETERS: 1,1,110:
            2,1,42;

COUNTERS:1,PARTS;

TALLIES:1,TIME IN SYSTEM:
        2,TIME ON CONVEYOR;

DSTATS:1,NR(C1)+NR(C2)+NR(C0)+NQ(C1Q)+NQ(C2Q)+NQ(C0Q),C0NVEY_UTIL:
       2,NR(STA0)+NQ(STA0Q),STA0_UTIL:
       3,NR(STA1)+NQ(STA1Q),STA1_UTIL:
       4,NR(STA2)+NQ(STA2Q),STA2_UTIL;

REPLICATE,10,,14400,NO,YES,7200;
```

```
          BEGIN;
          CREATE;

          ALTER:STA11,-1:
                STA22,-1:
                STA33,-1;
          ASSIGN:STA0BUS=0:
                 STA1BUS=0:
                 STA2BUS=0:
                 STA3BUS=0:
                 PIS=0:
                 MPIS=4:
                 INTERSTA=30:
                 DT=1:
                 STOPREQ=0:
                 NEXT(ENTRY);

ENTRY     ASSIGN:PIS=PIS+1:
                 EMPTY=0:
                 NEED0=1:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME):
                 NEXT(LSTA0);

LSTA0     BRANCH,1:
            IF,EMPTY==1,ELC1,YES:
            IF,(NEED0==1.AND.STA0BUS==1),DELENT,YES:
            IF,(NEED0==1.AND.STA0BUS==0),LSTA0A,YES:
            IF,(STA0BUS==1),                  LC1,YES:
            IF,(NEED1==1.OR.NEED2==1),        LC1,YES:
            IF,(NEED1==0.AND.NEED2==0),   LSTA0A,YES;


LSTA0A    QUEUE,STA0Q;
          SEIZE:STA0;
          ASSIGN:STA0BUS=1;
          BRANCH,2:
            ALWAYS,LSTA0C,YES:
            ALWAYS,STOPCON,NO;


LSTA0C    BRANCH,1:
            IF,((NEED1==0).AND.(NEED2==0)),DONEPART,YES:
            ELSE,LSTA0D,YES;

DONEPART  ASSIGN:TCT=TCT+(TNOW-ARCONV);
          DELAY:AINT(UNIF(40,60));
          COUNT:PARTS,1;
          TALLY:1,INT(ARTIME);
          TALLY:2,TCT;
          ASSIGN:EMPTY=0:
                 NEED0=0:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME);

LSTA0D    DELAY:AINT(UNIF(40,60));
```

```
          RELEASE:STA0;
          ASSIGN:STA0BUS=0:
                 NEED0=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC1,YES:
            ALWAYS,GOCON,NO;

LSTA1     BRANCH,2:
            IF,PIS<MPIS,ENTRY,NO:
            ALWAYS,LSTA1A,YES;

LSTA1A    BRANCH,1:
            IF,(EMPTY==1.AND.STA1BUS==0),ELC2,YES:
            IF,(EMPTY==0.AND.STA1BUS==1),LC2,YES:
            IF,(EMPTY==0.AND.STA1BUS==0),LSTA1B,YES:
            IF,(EMPTY==1.AND.STA1BUS==1),DSTA1,YES;

LSTA1B    BRANCH,1:
            IF,NEED1==1,LSTA1C,YES:
            ELSE,LC2,YES;

LSTA1C    QUEUE,STA1Q;
          SEIZE:STA1;
          ASSIGN:STA1BUS=1:
                 STA1DON=0:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED1=0;
          BRANCH,2:
            ALWAYS,LSTA1D,YES:
            ALWAYS,STOPCON,NO;

LSTA1D    DELAY:AINT(UNIF(40,60));
          BRANCH,3:
            ALWAYS,LSTA1E,YES:
            ALWAYS,EMTO2,NO:
            ALWAYS,GOCON,NO;

EMTO2     ASSIGN:EMPTY=1:NEXT(ELC2);

LSTA1E    DELAY:AINT(NORM(200,10));
          ASSIGN:STA1DON=1;
          QUEUE,STA11Q;
          SEIZE:STA11;
          BRANCH,2:
            ALWAYS,LSTA1F,YES:
            ALWAYS,STOPCON,NO;

LSTA1F    DELAY:AINT(UNIF(40,60));
          RELEASE:STA1;
          RELEASE:STA11;
          ALTER:STA11,-1;
          ASSIGN:STA1BUS=0:
                 STA1DON=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC2,YES:
            ALWAYS,GOCON,NO;

DSTA1     BRANCH,1:
```

```
            IF,STA1DON==1,DSTA1A,YES:
            ELSE,DSTA1B,YES;

DSTA1A      ALTER:STA11,1:DISPOSE;

DSTA1B      BRANCH,1:
              ALWAYS,ELC2,YES;

LSTA2       BRANCH,1:
              IF,(EMPTY==1.AND.STA2BUS==0),ELC3,YES:
              IF,(EMPTY==0.AND.STA2BUS==1),LC3,YES:
              IF,(EMPTY==0.AND.STA2BUS==0),LSTA2B,YES:
              IF,(EMPTY==1.AND.STA2BUS==1),DSTA2,YES;

LSTA2B      BRANCH,1:
              IF,NEED2==1,LSTA2C,YES:
              ELSE,LC3,YES;

LSTA2C      QUEUE,STA2Q;
            SEIZE:STA2;
            ASSIGN:STA2BUS=1:
                   STA2DON=0:
                   TCT=TCT+(TNOW-ARCONV):
                   NEED2=0;
            BRANCH,2:
              ALWAYS,LSTA2D,YES:
              ALWAYS,STOPCON,NO;

LSTA2D      DELAY:AINT(UNIF(40,60));
            BRANCH,3:
              ALWAYS,LSTA2E,YES:
              ALWAYS,EMTO3,NO:
              ALWAYS,GOCON,NO;

EMTO3       ASSIGN:EMPTY=1:NEXT(ELC3);

LSTA2E      DELAY:AINT(TRIA(500,525,600));
            ASSIGN:STA2DON=1;
            QUEUE,STA22Q;
            SEIZE:STA22;
            BRANCH,2:
              ALWAYS,LSTA2F,YES:
              ALWAYS,STOPCON,NO;

LSTA2F      DELAY:AINT(UNIF(40,60));
            RELEASE:STA2;
            RELEASE:STA22;
            ALTER:STA22,-1;
            ASSIGN:STA2BUS=0:
                   STA2DON=0:
                   MARK(ARCONV);
            BRANCH,2:
              ALWAYS,LC3,YES:
              ALWAYS,GOCON,NO;

DSTA2       BRANCH,1:
              IF,STA2DON==1,DSTA2A,YES:
              ELSE,DSTA2B,YES;

DSTA2A      ALTER:STA22,1:DISPOSE;
```

```
DSTA2B    BRANCH,1:
            ALWAYS,ELC3,YES;

LSTA3     BRANCH,1:
            IF,(EMPTY==1.AND.STA3BUS==0),ELC0,YES:
            IF,(EMPTY==0.AND.STA3BUS==1),LC0,YES:
            IF,(EMPTY==0.AND.STA3BUS==0),LSTA3B,YES:
            IF,(EMPTY==1.AND.STA3BUS==1),DSTA3,YES;

LSTA3B    BRANCH,1:
            IF,NEED2==1,LSTA3C,YES:
            ELSE,LC0,YES;

LSTA3C    QUEUE,STA3Q;
          SEIZE:STA3;
          ASSIGN:STA3BUS=1:
                 STA3DON=0:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED2=0;
          BRANCH,2:
            ALWAYS,LSTA3D,YES:
            ALWAYS,STOPCON,NO;

LSTA3D    DELAY:AINT(UNIF(40,60));
          BRANCH,3:
            ALWAYS,LSTA3E,YES:
            ALWAYS,EMTO0,NO:
            ALWAYS,GOCON,NO;

EMTO0     ASSIGN:EMPTY=1:NEXT(ELC0);

LSTA3E    DELAY:AINT(TRIA(500,525,600));
          ASSIGN:STA3DON=1;
          QUEUE,STA33Q;
          SEIZE:STA33;
          BRANCH,2:
            ALWAYS,LSTA3F,YES:
            ALWAYS,STOPCON,NO;

LSTA3F    DELAY:AINT(UNIF(40,60));
          RELEASE:STA3;
          RELEASE:STA33;
          ALTER:STA33,-1;
          ASSIGN:STA3BUS=0:
                 STA3DON=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC0,YES:
            ALWAYS,GOCON,NO;

DSTA3     BRANCH,1:
            IF,STA3DON==1,DSTA3A,YES:
            ELSE,DSTA3B,YES;

DSTA3A    ALTER:STA33,1:DISPOSE;

DSTA3B    BRANCH,1:
            ALWAYS,ELC0,YES;
```

```
LC1       ASSIGN:CON1T=0;
R1        QUEUE,C1Q;
          SEIZE:C1;
          DELAY:DT;
          ASSIGN:CON1T=CON1T+1;
          RELEASE:C1;
          BRANCH,1:
            IF,CON1T==INTERSTA,LSTA1,YES:
            ELSE,R1,YES;

ELC1      ASSIGN:CON1T=0;
ER1       QUEUE,EC1Q;
          SEIZE:EC1;
          DELAY:DT;
          ASSIGN:CON1T=CON1T+1;
          RELEASE:EC1;
          BRANCH,1:
            IF,CON1T==INTERSTA,LSTA1,YES:
            ELSE,ER1,YES;

LC2       ASSIGN:CON2T=0;
R2        QUEUE,C2Q;
          SEIZE:C2;
          DELAY:DT;
          ASSIGN:CON2T=CON2T+1;
          RELEASE:C2;
          BRANCH,1:
            IF,CON2T==INTERSTA,LSTA2,YES:
            ELSE,R2,YES;

ELC2      ASSIGN:CON2T=0;
ER2       QUEUE,EC2Q;
          SEIZE:EC2;
          DELAY:DT;
          ASSIGN:CON2T=CON2T+1;
          RELEASE:EC2;
          BRANCH,1:
            IF,CON2T==INTERSTA,LSTA2,YES:
            ELSE,ER2,YES;

LC3       ASSIGN:CON3T=0;
R3        QUEUE,C3Q;
          SEIZE:C3;
          DELAY:DT;
          ASSIGN:CON3T=CON3T+1;
          RELEASE:C3;
          BRANCH,1:
            IF,CON3T==INTERSTA,LSTA3,YES:
            ELSE,R3,YES;

ELC3      ASSIGN:CON3T=0;
ER3       QUEUE,EC3Q;
          SEIZE:EC3;
          DELAY:DT;
          ASSIGN:CON3T=CON3T+1;
          RELEASE:EC3;
          BRANCH,1:
            IF,CON3T==INTERSTA,LSTA3,YES:
            ELSE,ER3,YES;
```

```
LC0       ASSIGN:CON0T=0;
R0        QUEUE,C0Q;
          SEIZE:C0;
          DELAY:DT;
          ASSIGN:CON0T=CON0T+1;
          RELEASE:C0;
          BRANCH,1:
            IF,CON0T==INTERSTA,LSTA0,YES:
            ELSE,R0,YES;

ELC0      ASSIGN:CON0T=0;
ER0       QUEUE,EC0Q;
          SEIZE:EC0;
          DELAY:DT;
          ASSIGN:CON0T=CON0T+1;
          RELEASE:EC0;
          BRANCH,1:
            IF,CON0T==INTERSTA,LSTA0,YES:
            ELSE,ER0,YES;

GOCON     BRANCH,1:
            IF,STOPREQ==1,ENABLE,YES:
            ELSE,DONT,YES;

ENABLE    ALTER:C1,MPIS:
                C2,MPIS:
                C3,MPIS:
                C0,MPIS:
                EC1,MPIS:
                EC2,MPIS:
                EC3,MPIS:
                EC0,MPIS;

DONT      ASSIGN:STOPREQ=STOPREQ-1:
                 DISPOSE;

STOPCON   BRANCH,1:
            IF, STOPREQ==0,DISABLE,YES:
            ELSE,DONOT,YES;

DISABLE   ALTER:C1,-MPIS:
                C2,-MPIS:
                C3,-MPIS:
                C0,-MPIS:
                EC1,-MPIS:
                EC2,-MPIS:
                EC3,-MPIS:
                EC0,-MPIS;

DONOT     ASSIGN:STOPREQ=STOPREQ+1:
                 DISPOSE;

DELENT    DELAY:DT:NEXT(LSTA0);
```

```
BEGIN;
DISCRETE,,11,15;
PROJECT,TC202,Don Anderson;


ATTRIBUTES:ARCONV:
           ARTIME:
           EMPTY:
           NEED0:
           NEED1:
           NEED2:
           CON1T:
           CON2T:
           CON3T:
           CON0T:
           TCT;

RESOURCES: STA0:
           C1,4:
           EC1,4:
           STA1:
           STA11:
           C2,4:
           EC2,4:
           STA2:
           STA22:
           C3,4:
           EC3,4:
           STA3:
           STA33:
           C0,4:
           EC0,4;

QUEUES:    STA0Q:
           C1Q:
           EC1Q:
           STA1Q:
           STA11Q:
           C2Q:
           EC2Q:
           STA2Q:
           STA22Q:
           C3Q:
           EC3Q:
           STA3Q:
           STA33Q:
           C0Q:
           EC0Q;

VARIABLES: STA0BUS:
           STA1BUS:
           STA2BUS:
           STA3BUS:
           STA1DON:
           STA2DON:
           STA3DON:
           DT:
           INTERSTA:
           STOPREQ:
           PIS:
```

```
            MPIS;

PARAMETERS: 1,1,110:
             2,1,42;

COUNTERS:1,PARTS;

TALLIES:1,TIME IN SYSTEM:
        2,TIME ON CONVEYOR;

DSTATS:1,NR(C1)+NR(C2)+NR(C3)+NR(C0)+
         NQ(C1Q)+NQ(C2Q)+NQ(C3Q)+NQ(C0Q),CONVEY_UTIL:
       2,NR(STA0)+NQ(STA0Q),STA0_UTIL:
       3,NR(STA1)+NQ(STA1Q),STA1_UTIL:
       4,NR(STA2)+NQ(STA2Q),STA2_UTIL:
       5,NR(STA3)+NQ(STA3Q),STA3_UTIL;

REPLICATE,10,,14400,NO,YES,7200;
```

```
          BEGIN;
          CREATE;

          ASSIGN:STA0BUS=0:
                 STA1BUS=0:
                 STA2BUS=0:
                 STA3BUS=0:
                 PIS=0:
                 MPIS=4:
                 INTERSTA=30:
                 DT=1:
                 STOPREQ=0:
                 NEXT(ENTRY);

ENTRY     ASSIGN:PIS=PIS+1:
                 NEED0=1:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME):
                 NEXT(LSTA0);

LSTA0     BRANCH,1:
            IF,(NEED0==1.AND.STA0BUS==1),DELENT,YES:
            IF,(NEED0==1.AND.STA0BUS==0),LSTA0A,YES:
            IF,(STA0BUS==1),                LC1,YES:
            IF,(NEED1==1.OR.NEED2==1),      LC1,YES:
            IF,(NEED1==0.AND.NEED2==0),  LSTA0A,YES;

LSTA0A    QUEUE,STA0Q;
          SEIZE:STA0;
          ASSIGN:STA0BUS=1;

LSTA0C    BRANCH,1:
            IF,((NEED1==0).AND.(NEED2==0)),DONEPART,YES:
            ELSE,NEWPART,YES;

DONEPART  ASSIGN:TCT=TCT+(TNOW-ARCONV);
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,DP2,YES;

DP2       DELAY:10;
          BRANCH,2:
            ALWAYS,DP3,YES:
            ALWAYS,GOCON,NO;

DP3       DELAY:UNIF(40,60);
          COUNT:PARTS,1;
          TALLY:1,INT(ARTIME);
          TALLY:2,TCT;
          ASSIGN:NEED0=0:
                 NEED1=1:
                 NEED2=1:
                 TCT=0:
                 MARK(ARTIME);

NEWPART   DELAY:UNIF(40,60);
          BRANCH,2:
            ALWAYS,STOPCON,NO:
```

```
            ALWAYS,LSTA0E,YES;

LSTA0E      DELAY:10;
            RELEASE:STA0;
            ASSIGN:STA0BUS=0:
                   NEED0=0:
                   MARK(ARCONV);
            BRANCH,2:
              ALWAYS,LC1,YES:
              ALWAYS,GOCON,NO;

LSTA1       BRANCH,2:
              IF,PIS<MPIS,ENTRY,NO:
              ALWAYS,LSTA1A,YES;

LSTA1A      BRANCH,1:
              IF,STA1BUS==1,LC2,YES:
              ELSE,LSTA1B,YES;

LSTA1B      BRANCH,1:
              IF,NEED1==1,LSTA1C,YES:
              ELSE,LC2,YES;

LSTA1C      QUEUE,STA1Q;
            SEIZE:STA1;
            ASSIGN:STA1BUS=1:
                   TCT=TCT+(TNOW-ARCONV):
                   NEED1=0;
            BRANCH,2:
              ALWAYS,STOPCON,NO:
              ALWAYS,LSTA1D,YES;

LSTA1D      DELAY:10;
            BRANCH,2:
              ALWAYS,LSTA1E,YES:
              ALWAYS,GOCON,NO;

LSTA1E      DELAY:UNIF(40,60);
            DELAY:NORM(200,10);
            DELAY:UNIF(40,60);
            BRANCH,2:
              ALWAYS,STOPCON,NO:
              ALWAYS,LSTA1F,YES;

LSTA1F      DELAY:10;
            RELEASE:STA1;
            ASSIGN:STA1BUS=0:
                   MARK(ARCONV);
            BRANCH,2:
              ALWAYS,LC2,YES:
              ALWAYS,GOCON,NO;

LSTA2       BRANCH,1:
              IF,STA2BUS==1,LC3,YES:
              ELSE,LSTA2B,YES;

LSTA2B      BRANCH,1:
              IF,NEED2==1,LSTA2C,YES:
              ELSE,LC3,YES;
```

```
LSTA2C    QUEUE,STA2Q;
          SEIZE:STA2;
          ASSIGN:STA2BUS=1:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED2=0;
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA2D,YES;

LSTA2D    DELAY:10;
          BRANCH,2:
            ALWAYS,LSTA2E,YES:
            ALWAYS,GOCON,NO;

LSTA2E    DELAY:UNIF(40,60)42;
          DELAY:TRIA(500,525,600);
          DELAY:UNIF(40,60);
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA2F,YES;

LSTA2F    DELAY:10;
          RELEASE:STA2;
          ASSIGN:STA2BUS=0:
                 MARK(ARCONV);
          BRANCH,2:
            ALWAYS,LC3,YES:
            ALWAYS,GOCON,NO;

LSTA3     BRANCH,1:
            IF,STA3BUS==1,LC0,YES:
            ELSE,LSTA3B,YES;

LSTA3B    BRANCH,1:
            IF,NEED2==1,LSTA3C,YES:
            ELSE,LC0,YES;

LSTA3C    QUEUE,STA3Q;
          SEIZE:STA3;
          ASSIGN:STA3BUS=1:
                 TCT=TCT+(TNOW-ARCONV):
                 NEED2=0;
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA3D,YES;

LSTA3D    DELAY:10;
          BRANCH,2:
            ALWAYS,LSTA3E,YES:
            ALWAYS,GOCON,NO;

LSTA3E    DELAY:UNIF(40,60);
          DELAY:TRIA(500,525,600);
          DELAY:UNIF(40,60);
          BRANCH,2:
            ALWAYS,STOPCON,NO:
            ALWAYS,LSTA3F,YES;

LSTA3F    DELAY:10;
          RELEASE:STA3;
```

```
            ASSIGN:STA3BUS=0:
                   MARK(ARCONV);
            BRANCH,2:
              ALWAYS,LC0,YES:
              ALWAYS,GOCON,NO;

LC1         ASSIGN:CON1T=0;
R1          QUEUE,C1Q;
            SEIZE:C1;
            DELAY:DT;
            ASSIGN:CON1T=CON1T+1;
            RELEASE:C1;
            BRANCH,1:
              IF,CON1T==INTERSTA,LSTA1,YES:
              ELSE,R1,YES;

LC2         ASSIGN:CON2T=0;
R2          QUEUE,C2Q;
            SEIZE:C2;
            DELAY:DT;
            ASSIGN:CON2T=CON2T+1;
            RELEASE:C2;
            BRANCH,1:
              IF,CON2T==INTERSTA,LSTA2,YES:
              ELSE,R2,YES;

LC3         ASSIGN:CON3T=0;
R3          QUEUE,C3Q;
            SEIZE:C3;
            DELAY:DT;
            ASSIGN:CON3T=CON3T+1;
            RELEASE:C3;
            BRANCH,1:
              IF,CON3T==INTERSTA,LSTA3,YES:
              ELSE,R3,YES;

LC0         ASSIGN:CON0T=0;
R0          QUEUE,C0Q;
            SEIZE:C0;
            DELAY:DT;
            ASSIGN:CON0T=CON0T+1;
            RELEASE:C0;
            BRANCH,1:
              IF,CON0T==INTERSTA,LSTA0,YES:
              ELSE,R0,YES;
GOCON       BRANCH,1:
              IF,STOPREQ==1,ENABLE,YES:
              ELSE,DONT,YES;

ENABLE      ALTER:C1,MPIS:
                  C2,MPIS:
                  C3,MPIS:
                  C0,MPIS;

DONT        ASSIGN:STOPREQ=STOPREQ-1:
                   DISPOSE;

STOPCON     BRANCH,1:
              IF, STOPREQ==0,DISABLE,YES:
              ELSE,DONOT,YES;
```

```
DISABLE  ALTER:C1,-MPIS:
               C2,-MPIS:
               C3,-MPIS:
               C0,-MPIS;

DONOT    ASSIGN:STOPREQ=STOPREQ+1:
                DISPOSE;

DELENT   DELAY:DT:NEXT(LSTA0);
```

```
BEGIN;
DISCRETE,,11,8;
PROJECT,TC302,Don Anderson;


ATTRIBUTES:ARCONV:
           ARTIME:
           EMPTY:
           NEED0:
           NEED1:
           NEED2:
           CON1T:
           CON2T:
           CON3T:
           CON0T:
           TCT;

RESOURCES: STA0:
           C1,4:
           STA1:
           C2,4:
           STA2:
           C3,4:
           STA3:
           C0,4;

QUEUES:    STA0Q:
           C1Q:
           STA1Q:
           C2Q:
           STA2Q:
           C3Q:
           STA3Q:
           C0Q;

VARIABLES: STA0BUS:
           STA1BUS:
           STA2BUS:
           STA3BUS:
           DT:
           INTERSTA:
           STOPREQ:
           PIS:
           MPIS;

PARAMETERS: 1,1,110:
            2,1,42;

COUNTERS:1,PARTS;

TALLIES:1,TIME IN SYSTEM:
        2,TIME ON CONVEYOR;

DSTATS:1,NR(C1)+NR(C2)+NR(C3)+NR(C0)+
         NQ(C1Q)+NQ(C2Q)+NQ(C3Q)+NQ(C0Q),CONVEY_UTIL:
       2,NR(STA0)+NQ(STA0Q),STA0_UTIL:
       3,NR(STA1)+NQ(STA1Q),STA1_UTIL:
       4,NR(STA2)+NQ(STA2Q),STA2_UTIL:
       5,NR(STA3)+NQ(STA3Q),STA3_UTIL;
```

```
REPLICATE,10,,14400,NO,YES,7200;
```

6.2 Appendix B

The CIMS lab FMC Simulation Software Source Code

```
////////////////////////////////////////////////////////////////  03/30/96
#include <stdlib.h>
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <ctype.h>

const int true=1;
const int false=0;


//////////////////////////////////////////////////////////// DATA STRUCTURES


struct event
  {
   int isempty;
   char name[10];
   int stanum;
   long int time;
   long int stime;
   int partnum;
   long int artime;
   long int arconv;
   int totconvtime;
   int inorder;
   char need[11][13];
   struct event* next;
  };


struct stations
  {
   int busy;
   int done;
   char name[11];
   int conveyorwait;
   int palletwait;
   char robot_time[5];
     int rtp1;
     int rtp2;
     int rtp3;
     int rtp4[10];
     int rtp5[10];
   char service_time[5];
     int stp1;
     int stp2;
     int stp3;
     int stp4[10];
     int stp5[10];
  };


//////////////////////////////////////////////////////////// GLOBAL VARIABLES

int         rep_num,
            num_machines,
            inorder,
```

```
            parts_in_system,
            max_parts_in_system,
            pallets_in_system,
            max_pallets_in_system,
            separate_unloader,
            delayed_entry,
            length_of_conveyor,
            speed_of_conveyor,
            outfile,
            debug,
            debug_u,
            debug_w,
            trace,
            reps;

long int    interstation_time,
            pallet_unload_time,
            tnow,
            frozen,
            freeze_till_time,

            total_time,
            total_time_sq,
            total_t_o_c,
            total_t_o_c_sq,

            length_of_sim,
            sim_end_time,
            warm_up,

            part_num,
            total_parts,
            parts_com,

            time_in_syst,
            min_t_i_s,
            max_t_i_s,

            time_on_conveyor,
            min_t_o_c,
            max_t_o_c;

double      avg_time_syst,
            avg_t_o_c,

            mr_p_c,
            mr_p_c_sq,

            mr_t_i_s,
            mr_t_i_s_sq,

            mr_t_o_c,
            mr_t_o_c_sq,
            mr_util_conv,

            fl_length_of_sim,
            util_conv,
            util_sta[11],
            mr_util_sta[11];
```

```
char        mach_seq_reqd[11][13];

char        first_name[15];
char        last_name[15];
char        date[15];
char        confile_name[25];
char        outfile_name[25];

int         outused[6];

struct stations station[11];

event *     calender1;
event *     calender2;
event *     current;
event *     temp_storage[11];

FILE *     fp;
FILE *     fpc;
FILE *     fpt;
FILE *     fpdb;

//////////////////////////////////////////////////////// FUNCTION PROTOTYPES

int         main_menu_config(void);
int         main_menu_simulat(void);
int         main_menu_output(void);

int         test_validity_of_parts_req(void);
void        change_number_pal(void);
void        change_load_sta(void);
void        change_number_machines(void);
void        change_machine_params(int);
void        change_remove_sta(void);
void        change_machining_seq(void);
void        change_number_reps(void);
void        change_length_conveyor(void);
void        change_speed_conveyor(void);
void        change_pallet_unload(void);
void        change_length_sim(void);
void        change_warm_up(void);
void        change_station_conveyor(int);
void        change_station_name(int);
void        change_order(void);
void        add_a_machine(void);
void        remove_a_machine(void);

void        initialize(void);
void        initialize_system(void);
void        init_for_sim(void);
void        init_for_rep(void);
void        read_confile(void);
void        write_confile(void);

void        testcase0(void);
void        testcase1(void);
void        testcase2(void);
void        testcase3(void);

void        reset_stats(void);
```

```
void        reset_mr_stats(void);
void        reset_utils(void);
void        reset_mr_utils(void);


void        update_stats(void);
void        update_utils(long int,long int);
void        update_mr_stats(void);
void        update_mr_utils(void);

event*      create_event (void);
void        start_one_thru(long int);
void        warm_it_up(void);

void        schedule_current(void);
event*      remove_next(void);
long int    get_next_time(void);

int         canservice(int stanum, struct stations sta);
long int    determine_robot_time(int);
long int    determine_service_time(int);
void        inc_stanum(void);

void        delay_parts_on_conveyor(long int t);
void        freeze_conveyor(long int t);

void        enter_system(void);
void        arrive_at_station(void);
void        leave_station(void);
void        leave_service_pw(void);
void        leave_s2(void);
void        unbusy_station(void);
void        emptygo_station(void);
void        done_station(void);

void        printcals(void);

void        print_rep_report(int);

void        print_overallreport(void);

void        display_current_config_scr(void);
void        print_current_config_file(void);

void        display_current_simulation_scr(void);
void        print_current_simulation_file(void);

void        display_current_operations_scr(void);
void        print_current_operations_file(void);

void        display_station_service(int);
void        display_station_robot(int);
void        display_intro_screen(void);

void        prompt_for_robot_distribution(int);
void        prompt_for_service_distribution(int);


int         read_integer(void);
int         digitcount(int);
```

```
void        print_the_distribution(void);

void        simulate_a_rep(void);

//////////////////////////////////////////////////////////////////// MAIN ROUTINE

void main(void)
{
 int i,menu,done,contin,f,g;
 char* waste;
 char resp[20];
 char c;

 trace=false;
 debug=false;
 //
 // trace=true;
 // fpt=fopen(sim0.trc);
 //
 if (trace) fprintf(fpt,"MAIN\n");

 outfile=false;
 initialize_system();

 display_intro_screen();

 testcase0();
 contin=true;

 while (contin)
   {
    initialize();
    menu=1;

    while (menu<=3)
      {
        if (menu==1)
            menu=main_menu_config();
        else if (menu==2)
            menu=main_menu_simulat();
        else if (menu==3)
            menu=main_menu_output();
      }

    if (menu!=5)
     {
       init_for_sim();

       if(outfile)
        { print_current_config_file();
          print_current_simulation_file();
        }

       start_one_thru(tnow);
       pallets_in_system=1;

       warm_it_up();

       print_rep_report(0);
```

```
        printf("press enter for next screen: ");
        gets(resp);

        sim_end_time=warm_up+length_of_sim;


        for (rep_num=1;rep_num<=reps;rep_num++)
         {
            init_for_rep();
            simulate_a_rep();

            print_rep_report(rep_num);

            printf("press enter for next screen: ");
            gets(resp);

            update_mr_stats();
            update_mr_utils();

            sim_end_time+=length_of_sim;
         }

        if (reps > 1)
         {
          clrscr();
          print_overallreport();
          printf("\n press enter for next screen: ");
          gets(resp);
         }
      }
     else
      {
         contin=false;
         if (outfile) fclose(fp);
         if (trace) fclose(fpt);
         if (debug) fclose(fpdb);

      }

    }

}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                      SYSTEM INITIALIZATION MODULES                 //
//                                                                    //
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////// INITIALIZE_SYSTEM

void initialize_system(void)
{
 int i,j;

 for (i=0; i<11; i++)
```

```
    { station[i].busy=false;
      station[i].done=false;
      strcpy(station[i].name,"remove");
      station[i].conveyorwait=true;
      station[i].palletwait=false;

      strcpy(station[i].robot_time,"norm");
      station[i].rtp1=0;
      station[i].rtp2=0;
      station[i].rtp3=0;
      for (j=0; j<10; j++)
        {
          station[i].rtp4[j]=0;
          station[i].rtp5[j]=0;
        }
      strcpy(station[i].service_time,"tria");
      station[i].stp1=0;
      station[i].stp2=0;
      station[i].stp3=0;
      for (j=0; j<10; j++)
        {
          station[i].stp4[j]=0;
          station[i].stp5[j]=0;
        }
      strcpy(mach_seq_reqd[i],"remove");
    }

  length_of_sim=2000;                          //20;
  fl_length_of_sim=(float)length_of_sim;

  warm_up=700;                                 //72;

  reps=1;
  max_pallets_in_system=2;

 }

 ////////////////////////////////////////////////////////////////// INITIALIZE

 void initialize(void)
 {
   int i;

   if (trace) fprintf(fp,"INITIALIZE\n");

   tnow=0;
   parts_in_system=0;
   pallets_in_system=0;

   calender1 = create_event();
   calender2 = create_event();

   part_num=1;
   delayed_entry=0;
   frozen=false;
   freeze_till_time=0;

   for (i=0; i<11; i++)
     { station[i].busy=false;
       station[i].done=false;
```

```
    }
}

////////////////////////////////////////////////////////////////// INIT FOR SIM

void init_for_sim(void)
{
  if (trace) fprintf(fp,"INIT_FOR_SIM\n");

  reset_stats();
  reset_mr_stats();
  reset_utils();
  reset_mr_utils();

  fl_length_of_sim=(float)length_of_sim;
  interstation_time=(speed_of_conveyor/(num_machines+1+separate_unloader));


}

////////////////////////////////////////////////////////////////// INIT FOR REP

void init_for_rep(void)
{
  if (trace) fprintf(fp,"INIT_FOR_REP\n");

  reset_stats();
  reset_utils();

}

////////////////////////////////////////////////////////////////// WARM_IT_UP

void warm_it_up(void)
{
  int done;
  long int next_event_time;
  event* t;

  if (trace) fprintf(fp,"WARM_IT_UP\n");

  done=false;
  while (!done)
    {
      next_event_time=get_next_time();

      if (next_event_time >= warm_up)
        {
          update_utils(warm_up,tnow);
          tnow=warm_up;
          done=true;
        }
      else
        {
           update_utils(next_event_time,tnow);

           current=remove_next();
           t=current;

           if (debug)
```

```
            { fprintf(fpdb,"NEXT EVENT:\t");
              fprintf(fpdb,"partnum: %d\tname: %s\tsta: %d\ttime: %ld\n\n",
                           t->partnum,t->name,t->stanum,t->time);
            }

          tnow=current->time;

          if (tnow >= freeze_till_time)
              frozen=false;

          if (strcmp(current->name,"entersys") ==0)
              enter_system();

          else if(strcmp(current->name,"arrive") ==0)
              arrive_at_station();

          else if(strcmp(current->name,"unbusysta") ==0)
              unbusy_station();

          else if(strcmp(current->name,"donesta") ==0)
              done_station();

          else if(strcmp(current->name,"emptygo") ==0)
              emptygo_station();

          else if(strcmp(current->name,"leavesta") ==0)
              leave_station();

          else if(strcmp(current->name,"leaveserv") ==0)
              leave_service_pw();

          else if(strcmp(current->name,"leaves2") ==0)
              leave_s2();

          else if (outfile)
              fprintf(fp,"UNKNOWN CURRENT->NAME IN WARM_UP()\n");
        }
      if (debug)
        {
          printcals();
          fprintf(fpdb,"NEXT TIME: %ld\n",
                       next_event_time);

        }
    }
}

//////////////////////////////////////////////////////////// SIMULATE_A_REP

void simulate_a_rep(void)
{
  int      done;
  long int next_event_time;
  event*   t;

  if (trace) fprintf(fp,"SIMULATE_A_REP\n");

  done=false;

  while (!done)
```

```
{
   next_event_time=get_next_time();

   if (next_event_time >= sim_end_time)
     {
       update_utils(sim_end_time,tnow);
       tnow=sim_end_time;
       done=true;
     }
   else
     { update_utils(next_event_time,tnow);

       current=remove_next();
       t=current;

       if (debug)
         { fprintf(fpdb,"NEXT EVENT:\t");
           fprintf(fpdb,"partnum: %d\tname: %s\tsta: %d\ttime: %ld\n\n",
                              t->partnum,t->name,t->stanum,t->time);
         }

       tnow=current->time;

       if (tnow >= freeze_till_time)
           frozen=false;

       if (strcmp(current->name,"entersys") ==0)
           enter_system();

       else if(strcmp(current->name,"arrive") ==0)
           arrive_at_station();

       else if(strcmp(current->name,"unbusysta") ==0)
           unbusy_station();

       else if(strcmp(current->name,"donesta") ==0)
           done_station();

       else if(strcmp(current->name,"emptygo") ==0)
           emptygo_station();

       else if(strcmp(current->name,"leavesta") ==0)
           leave_station();

       else if(strcmp(current->name,"leaveserv") ==0)
           leave_service_pw();

       else if(strcmp(current->name,"leaves2") ==0)
           leave_s2();

       else if (outfile)
           fprintf(fp,"UNKNOWN CURRENT->NAME IN SIM_A_REP()\n");
     }
   if (debug)
     {
       printcals();
       fprintf(fpdb,"NEXT TIME: %ld\n",
                         next_event_time);

     }
```

```
    }
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                        USER INTERFACE MODULES                      //
//                                                                    //
////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////// INTRO_SCREEN

void display_intro_screen(void)
{
 char* waste;
 char resp[20];

 if (trace) fprintf(fp,"INTRO_SCREEN\n");

 clrscr();
 printf("\n\n\n\n\n\t\t  Computer Integrated Manufacturing Systems'\n\n\n");
 printf("\t\tFLEXIBLE MANUFACTURING CELL SIMULATION SYSTEM\n\n\n\t ");
 printf("_____________________________________________________\n\n\n");

 printf("\tPlease type your first name (maximum 15 characters): ");
 gets(first_name);
 if (strlen(first_name) > 15 )
   {
     printf("\tlength must not exceed 15 characters, please re-enter: ");
     gets(first_name);
   }

 printf("\tPlease type your last name (maximum 15 characters): ");
 gets(last_name);
 if (strlen(last_name) > 15 )
   {
     printf("\tlength must not exceed 15 characters, please re-enter: ");
     gets(last_name);
   }
 printf("\tPlease type today's date (mm/dd/yy): ");
 gets(date);
 if (strlen(date) > 15 )
   {
     printf("\tlength must not exceed 8 characters, please re-enter: ");
     gets(date);
   }
}

////////////////////////////////////////////////////////// MAIN MENU CONFIG

int main_menu_config(void)
{
 int tc,cha,dud,sat,ret;
 int opt;
 char resp[20];
 char* waste;
```

```
if (trace) fprintf(fpt,"MAIN_MENU_CONFIG\n");

sat=false;

while (!sat)
  {
    clrscr();
    printf ("\t\tMAIN MENU FOR SYSTEM CONFIGURATION\n\n");

    display_current_config_scr();

    printf("\nEnter a line number to edit that line's information \n");
    printf("  or to invoke the desired option: ");
    opt=read_integer();

    while ( (opt<0 || opt>num_machines+9)&&
            (opt!=100&&opt!=111&&opt!=222&&opt!=333) )
      {
        printf("\nThe response entered is not valid.  Please choose");
        printf(" from the displayed\n line numbers (line numbers are ");
        printf("in < > brackets): ");
        opt=read_integer();
      }


    if (opt == 0)                          change_load_sta();
    else if (opt <= num_machines)          change_machine_params(opt);
    else if (opt==num_machines+1)          change_remove_sta();
    else if (opt==num_machines+2)          change_length_conveyor();
    else if (opt==num_machines+3)          change_speed_conveyor();
    else if (opt==num_machines+4)          change_pallet_unload();
    else if (opt==num_machines+5)          add_a_machine();
    else if (opt==num_machines+6)          remove_a_machine();
    else if (opt==num_machines+7)
      {
        printf("enter a name for the configuration file: ");
        gets(confile_name);
        fpc=fopen(confile_name,"r");
        if (fpc!='\0')
          {
            read_confile();
            fclose(fpc);
          }
        else
          {
            printf("file not found....     press enter");
            gets(resp);
          }
      }
    else if (opt==num_machines+8)    {
                                       sat=true;
                                       ret=2;
                                     }
    else if (opt==num_machines+9)    {
                                       sat=true;
                                       ret=5;
                                     }
    else if (opt==100)                 testcase0();
    else if (opt==111)                 testcase1();
```

```
      else if (opt==222)                testcase2();
      else if (opt==333)                testcase3();


    }

 return ret;
}

////////////////////////////////////////////////// DISPLAY CURRENT CONFIG SCR

void display_current_config_scr(void)
{
 int i,j,c,done,size,ind;
 int done_user,index,perc;

 if (trace) fprintf(fpt,"PRINT_CURRENT_CONFIG_SCR\n");

 //clrscr();
 printf("Station  Station  Conveyor  Robot time                Service time\n");
 printf("number   name     /Pallet   distribution              distribution\n");
 printf("                  behavior  (in seconds)              (in seconds)\n");
 printf("_______  _______  ________  ____________              _____________\n");

 for (i=0; i<=num_machines+separate_unloader; i++)
   {
     if (i<10)
       printf("<%d>  %d   %s",i,i,station[i].name);
     else
       printf("<%d> %d %s",i,i,station[i].name);

     size=strlen(station[i].name);
     for (c=1; c<=9-size; c++)
       printf(" ");
     if (!station[i].conveyorwait && !station[i].palletwait)

           printf("   C/C    ");

     else if (station[i].conveyorwait)

           printf("   W/W    ");
     else
           printf("   C/W    ");

     if (strcmp(station[i].robot_time,"tria")==0)
       {
           printf("%s(%d,%d,%d)",  station[i].robot_time,
                                   station[i].rtp1,
                                   station[i].rtp2,
                                   station[i].rtp3 );

           size=2+digitcount(station[i].rtp1)+digitcount(station[i].rtp2)+
                  digitcount(station[i].rtp3);
       }
     else if (strcmp(station[i].robot_time,"norm")==0 ||
              strcmp(station[i].robot_time,"unif")==0 )
       {
           printf("%s(%d,%d)",     station[i].robot_time,
                                   station[i].rtp1,
                                   station[i].rtp2);
```

```
        size=1+digitcount(station[i].rtp1)+digitcount(station[i].rtp2);
    }
  else if ( (strcmp(station[i].robot_time,"expo")==0) ||
            (strcmp(station[i].robot_time,"cons")==0) )
    {
        printf("%s(%d)",        station[i].robot_time,
                                station[i].rtp1);

        size=digitcount(station[i].rtp1);
    }
  else
    {    printf("%s(",           station[i].robot_time);

         done_user=false;
         index=0;
         perc=0;
         size=0;
         while ( !done_user )
           { printf("%d:%d",    station[i].rtp4[index],
                                station[i].rtp5[index]);

             perc+=station[i].rtp5[index];

             size++;
             size+=digitcount(station[i].rtp4[index]);
             size+=digitcount(station[i].rtp5[index]);

             if ( perc < 100 )
               {
                 printf(",");
                 size++;
               }
             else
               { done_user=true;
                 printf(")");
               }
             index++;
           }

   }

  for (c=1; c<=19-size; c++)
    printf(" ");

  if ( (strcmp(station[i].name,"enter")!=0) &&
       (strcmp(station[i].name,"remove")!=0) &&
       (strcmp(station[i].name,"exit" )!=0) )
    {

      if (strcmp(station[i].service_time,"tria")==0)

        printf("%s(%d,%d,%d)",station[i].service_time,
                              station[i].stp1,
                              station[i].stp2,
                              station[i].stp3 );

      else if (strcmp(station[i].service_time,"norm")==0 ||
           strcmp(station[i].service_time,"unif")==0 )
```

```
          printf("%s(%d,%d)",station[i].service_time,
                             station[i].stp1,
                             station[i].stp2);

        else if ( (strcmp(station[i].service_time,"expo")==0) ||
                  (strcmp(station[i].service_time,"cons")==0) )

          printf("%s(%d)",station[i].service_time,
                          station[i].stp1);
        else
          {
            printf("%s(",          station[i].service_time);

            done_user=false;
            index=0;
            perc=0;
            size=0;
            while ( !done_user )
              {
                printf("%d:%d",    station[i].stp4[index],
                                   station[i].stp5[index]);

                perc+=station[i].stp5[index];

                size++;
                size+=digitcount(station[i].stp4[index]);
                size+=digitcount(station[i].stp5[index]);

                if ( perc < 100 )
                  {
                   printf(",");
                   size++;
                  }
                else
                  { done_user=true;
                    printf(")");
                  }
                index++;
              }
          }
      }
    printf("\n");

  }
if (!separate_unloader)
  {
    ind=num_machines+1;
    if (num_machines==9)
      printf("<%d> %d  %s",ind,ind,station[ind].name);
    else
      printf("<%d>  %d   %s",ind,ind,station[ind].name);
    printf("    (the load station removes the finished parts)\n");
  }

ind=num_machines+2;
printf("_____________________________________________________\n");
printf("<%d>  Conveyor Length:\t%-d feet.\n",ind,length_of_conveyor);
printf("<%d>  Conveyor Speed: \t%-d seconds to complete 1 revolution.\n",
                                                ind+1, speed_of_conveyor);
printf("<%d>  Pallet load/unload time (C/W only):\t%-ld seconds.\n\n",ind+2,
```

```
                                                        pallet_unload_time);
  printf("<%d>  add a machine\n",ind+3);
  printf("<%d>  remove a machine\n\n",ind+4);
  printf("<%d>  load a configuration (previously saved) from file\n",ind+5);
  printf("<%d>  next screen\n",ind+6);
  printf("<%d>  quit\n",ind+7);
 }

 ////////////////////////////////////////////////////////////// READ INTEGER

 int read_integer(void)
 {
 int length,value,test,place,invalid,done;
 char resp[20];

  done=false;
  while (!done)
    {
      gets(resp);
      length=strlen(resp);
      value=0;
      invalid=false;
      if (length==0) invalid=true;
      for (place=0; place<length; place++)
       {
         test=isdigit(resp[place]);
         if (test==0)
            invalid=true;
         else
            value=10*value+(resp[place]-'0');
       }
      if (invalid)
         printf("only integer values are permitted. please re-enter: ");
      else done=true;
    }

  return value;
 }

 ////////////////////////////////////////////////////////////// READ CONFILE

 void read_confile(void)
 {
  int i,j;
  fscanf(fpc,"%d%d",&num_machines,&separate_unloader);
  for (i=0; i<=num_machines+separate_unloader; i++)
   {
     fscanf(fpc,"%s%d%d%s%d%d%d", station[i].name,
                                  &station[i].conveyorwait,
                                  &station[i].palletwait,
                                   station[i].robot_time,
                                  &station[i].rtp1,
                                  &station[i].rtp2,
                                  &station[i].rtp3);

     fscanf(fpc,"%s%d%d%d", station[i].service_time,
                            &station[i].stp1,
                            &station[i].stp2,
                            &station[i].stp3);
```

```
    for (j=0; j<10; j++)
      {
        fscanf(fpc,"%d",  &station[i].rtp4[j]);
        fscanf(fpc,"%d",  &station[i].rtp5[j]);
        fscanf(fpc,"%d",  &station[i].stp4[j]);
        fscanf(fpc,"%d",  &station[i].stp5[j]);
      }
  }
 fscanf(fpc,"%d%d%ld",&length_of_conveyor,&speed_of_conveyor,
                                          &pallet_unload_time);
 fscanf(fpc,"%d%d",&max_pallets_in_system,&inorder);
 for (i=0; i<11; i++)
     fscanf(fpc,"%s",mach_seq_reqd[i]);
 fscanf(fpc,"%d%ld%ld",&reps,&length_of_sim,&warm_up);
}

////////////////////////////////////////////////////////////// WRITE CONFILE

void write_confile(void)
{
 int i,j;

 fprintf(fpc,"%d\n%d\n",num_machines,separate_unloader);
 for (i=0; i<=num_machines+separate_unloader; i++)
  {
    fprintf(fpc,"%s\n%d\n%d\n%s\n%d\n%d\n%d\n",station[i].name,
                                               station[i].conveyorwait,
                                               station[i].palletwait,
                                               station[i].robot_time,
                                               station[i].rtp1,
                                               station[i].rtp2,
                                               station[i].rtp3);


    fprintf(fpc,"%s\n%d\n%d\n%d\n",station[i].service_time,
                                   station[i].stp1,
                                   station[i].stp2,
                                   station[i].stp3);


    for (j=0; j<10; j++)
      {
        fprintf(fpc,"%d\n",  station[i].rtp4[j]);
        fprintf(fpc,"%d\n",  station[i].rtp5[j]);
        fprintf(fpc,"%d\n",  station[i].stp4[j]);
        fprintf(fpc,"%d\n",  station[i].stp5[j]);
      }
  }
 fprintf(fpc,"%d\n%d\n%ld\n",length_of_conveyor,speed_of_conveyor,
                                                pallet_unload_time);
 fprintf(fpc,"%d\n%d\n",max_pallets_in_system,inorder);
 for (i=0; i<11; i++)
     fprintf(fpc,"%s\n",mach_seq_reqd[i]);
 fprintf(fpc,"%d\n%ld\n%ld\n\n",reps,length_of_sim,warm_up);
}

////////////////////////////////////////////////////// CHANGE LENGTH CONVEYOR

void change_length_conveyor(void)
```

```
{
int sat,numin,accept;
char resp[20];

 if (trace) fprintf(fpt,"CHANGE_LENGTH_CONVEYOR\n");


 accept=false;
 while (!accept)
   {
     printf("enter the new conveyor length in feet (integer, 1000 or less): ");
     numin=read_integer();
     if ( numin<=1000 )
       {
         length_of_conveyor=numin;
         accept=true;
       }
     else
       printf("invalid entry, please re-enter,\n");
   }

}

////////////////////////////////////////////////////// CHANGE SPEED CONVEYOR

void change_speed_conveyor(void)
{
int sat,numin,accept;
char resp[20];

 if (trace) fprintf(fpt,"CHANGE_SPEED_CONVEYOR\n");


 accept=false;
 while (!accept)
   {
     printf("enter the new conveyor speed (number of seconds for 1 ");
     printf("revolution)\n(integer, 10,000 or less): ");
     numin=read_integer();
     if ( (numin>=1)&&(numin<=10000) )
       {
         speed_of_conveyor=numin;
         accept=true;
       }
     else
       printf("invalid entry, please re-enter,\n");
   }

}
////////////////////////////////////////////////////// CHANGE PALLET UNLOAD

void change_pallet_unload(void)
{
int sat,numin,accept;
char resp[20];

 if (trace) fprintf(fpt,"CHANGE_PALLET_UNLOAD\n");


 accept=false;
```

```
  while (!accept)
    {
      printf("enter the pallet load/unload time (in seconds): ");
      numin=read_integer();
      if ( (numin>=1)&&(numin<=10000) )
        {
          pallet_unload_time=numin;
          accept=true;
        }
      else
        printf("invalid entry, range is (1,...,10,000), please re-enter,\n");
    }

 }

 ////////////////////////////////////////////////////////////// CHANGE LOAD STA

 void change_load_sta(void)
 {
  int opt;
  char resp[20];

  if (trace) fprintf(fpt,"CHANGE_LOAD_STA\n");


      clrscr();
      printf("\t\t\tCHANGE LOAD STATION\n\nThe load station");
      printf(" (station 0) is currently defined as follows:\n\n");
      printf("<->  Station name:\t\t%s\n",station[0].name);

      printf("<1>  Conveyor interaction:");

      if (station[0].conveyorwait)
        printf("\tThe Conveyor & Pallet both wait\n\n");
      else if (station[0].palletwait)
        printf("\tThe Conveyor continues, The Pallet waits\n\n");
      else
        printf("\tThe Conveyor continues, The Pallet continues\n\n");

      display_station_robot(0);

      printf("<3>  change everything \n");
      printf("<4>  accept current settings\n");
      printf("\nEnter a line number to edit that line's information \n");
      printf("  or to invoke the desired option: ");
      opt=read_integer();

      while (opt<1 || opt>4)
        {
          printf("\nThe response entered is not valid.  Please choose\n");
          printf(" from the displayed line numbers (line numbers are ");
          printf("in < > brackets: ");
          opt=read_integer();
        }

      if (opt==1)          change_station_conveyor(0);
      else if (opt==2)     prompt_for_robot_distribution(0);
      else if (opt==3) {   change_station_conveyor(0);
                           prompt_for_robot_distribution(0);
                       }
```

```
}

////////////////////////////////////////////////////// CHANGE STATION CONVEYOR

void change_station_conveyor(int stanum)
{
 int sat,accept;
 char resp[20];

 if (stanum<10)
    printf("\n\tSelect a station/conveyor interaction for station %d, %s: ",
                                          stanum,station[stanum].name);
 else
    printf("\n\tSelect a station/conveyor interaction for the new station:");

 printf("\n\n\t\t\t<1>: Conveyor & Pallet both wait\n\t\t\t<2>: Conveyor ");
 printf("continues, Pallet waits\n\t\t\t<3>: Conveyor ");
 printf("continues, Pallet continues (empty)\n\n\tselect from (1,2,3): ");
 gets(resp);

 while ( (strcmp(resp,"1")!=0)&&(strcmp(resp,"2")!=0)&&
         (strcmp(resp,"3")!=0) )
   {
     printf("you must select from (1,2,3), please re-enter: ");
     gets(resp);
   }

 if ( strcmp(resp,"1")==0 )
   { station[stanum].conveyorwait=true;
     station[stanum].palletwait=false;
   }
 else if ( strcmp(resp,"2")==0)
   { station[stanum].conveyorwait=false;
     station[stanum].palletwait=true;
   }
 else if ( strcmp(resp,"3")==0 )
   { station[stanum].conveyorwait=false;
     station[stanum].palletwait=false;
   }

}

////////////////////////////////////////////////////// CHANGE NUMBER MACHINES

void change_number_machines(void)                    // OBSOLETE?
{
int sat,numin,accept;
char resp[20];

 if (trace) fprintf(fpt,"CHANGE_NUMBER_MACHINES\n");

 sat=false;
 while (!sat)
   {
     clrscr();
     printf("\t\t\tCHANGE NUMBER OF MACHINES\n\n");
     printf("Currently, the number of machining stations specified is:\n\n");
     printf("\t\t\t %d \n\n",num_machines);
     printf("Is this satisfactory? (enter 1=yes 0=no): ");
       gets(resp);
```

```
    while ( (strcmp(resp,"1")!=0) && (strcmp(resp,"0")!=0) )
      {
        printf("you must select from (0,1), please re-enter: ");
          gets(resp);
      }
    if (strcmp(resp,"0")==0)
      {
        printf("enter the new number of machines (1,2,...,9): ");
        numin=read_integer();
        while((numin!=1)&&(numin!=2)&&(numin!=3)&&(numin!=4)&&(numin!=5)&&
              (numin!=6)&&(numin!=7)&&(numin!=8)&&(numin!=9))
          { printf("you must select from (1,2,3, ... ,9) please re-enter: ");
            numin=read_integer();
          }
        num_machines=numin;

      }
     else sat=true;
   }

}

////////////////////////////////////////////////////// CHANGE STATION NAME

void change_station_name(int stanum)
{
 int sat,lenn,accept;
 char resp[20];

 if (trace) fprintf(fpt,"CHANGE_MACHINE_NAME\n");

 accept=false;
 while (!accept)
   {
     if (stanum<10)
       {
         printf("\nenter a new name for station %d(max 9 characters): ",
                                                                 stanum);
         gets(resp);
       }
     else
       { printf("\nenter a name for the new station (max 9 characters): ");
         gets(resp);
       }

     lenn=strlen(resp);
     if (lenn<=9)
       {
         accept=true;
         strcpy(station[stanum].name,resp);
       }
     else
         printf("length must not exceed 9 characters, please re-enter: ");
   }
}

////////////////////////////////////////////////////// CHANGE MACHINE PARAMS

void change_machine_params(int mach)
{
```

```
int sat,opt;
char resp[20];

if (trace) fprintf(fpt,"CHANGE_MACHINE_PARAMS\n");

sat=false;
while (!sat)
  {
    clrscr();

    printf("\t\tCHANGE MACHINING STATION (station ");
    printf("%d)\n\n",mach);

    printf("<0>  Station name:\t\t%s\n\n",station[mach].name);
    printf("<1>  Conveyor interaction:");

    if      (station[mach].conveyorwait)
      printf("\tThe Conveyor & Pallet both wait\n\n");
    else if (station[mach].palletwait)
      printf("\tThe Conveyor continues, The Pallet waits\n\n");
    else
      printf("\tThe Conveyor continues, The Pallet continues\n\n");

    display_station_robot(mach);
    display_station_service(mach);

    printf("<4>  change everything \n");
    printf("<5>  accept current settings\n");
    printf("\nEnter a line number to edit that line's information \n");
    printf("  or to invoke the desired option: ");
    opt=read_integer();

    while ( opt<0 || opt>5)
      {
        printf("\nThe response entered is not valid.  Please choose\n");
        printf(" from the displayed line numbers (line numbers are ");
        printf("in < > brackets: ");
        opt=read_integer();
      }

    if (opt==0)           change_station_name(mach);
    else if (opt==1)      change_station_conveyor(mach);
    else if (opt==2)      prompt_for_robot_distribution(mach);
    else if (opt==3)      prompt_for_service_distribution(mach);
    else if (opt==4)    { change_station_name(mach);
                          change_station_conveyor(mach);
                          prompt_for_robot_distribution(mach);
                          prompt_for_service_distribution(mach);
                        }
    else sat=true;
  }
}

////////////////////////////////////////////////////////// CHANGE REMOVE STA

void change_remove_sta(void)
{
 int opt,i,stanum;
 char resp[20];
```

```
if (trace) fprintf(fpt,"CHANGE_REMOVE_STA\n");

printf("\t\t\tCHANGE REMOVEAL STATION\n\n");
stanum=num_machines+1;

if (separate_unloader)
  { clrscr();
    printf("\t\t\tCHANGE REMOVEAL STATION\n\nThe removal station");
    printf(" (station %d) is currently defined as follows:\n\n",stanum);
    printf("<->  Station name:\t%s\n",station[stanum].name);
    printf("<1>  Conveyor interaction:\t");

    if (station[stanum].conveyorwait)
      printf("\tThe Conveyor & Pallet both wait\n\n");
    else if (station[stanum].palletwait)
      printf("\tThe Conveyor continues, The Pallet waits\n\n");
    else
      printf("\tThe Conveyor continues, The Pallet continues\n\n");

    display_station_robot(stanum);
    printf("<3>  change everything \n");
    printf("<4>  accept current settings\n");
    printf("\nEnter a line number to edit that line's information \n");
    printf("  or to invoke the desired option: ");
    opt=read_integer();

    while (opt<1 || opt>4)
      {
        printf("\nThe response entered is not valid.  Please choose\n");
        printf(" from the displayed line numbers (line numbers are ");
        printf("in < > brackets: ");
        opt=read_integer();
      }

    if (opt==1)          change_station_conveyor(stanum);
    else if (opt==2)     prompt_for_robot_distribution(stanum);
    else if (opt==3) {   change_station_conveyor(stanum);
                         prompt_for_robot_distribution(stanum);
                     }

  }
else
  { clrscr();
    printf("\t\t\tCHANGE REMOVEAL STATION\n\n");
    printf("Currently, a separate unloader IS NOT specified. \n\n");
    printf("<1>  add removal station (duplicate enter station) \n");
    printf("<2>  add removal station (by specifying parameters) \n");
    printf("<3>  accept current setting \n");
    printf("\nEnter a line number to invoke the desired option: ");
    opt=read_integer();

    while (opt<1 || opt>3)
      {
        printf("\nThe response entered is not valid.  Please choose\n");
        printf(" from the displayed line numbers (line numbers are ");
        printf("in < > brackets: ");
        opt=read_integer();
      }
    if (opt==1)
      {
```

```
          separate_unloader=true;
          strcpy(station[stanum].name,"remove");
          station[stanum].conveyorwait=station[0].conveyorwait;
          station[stanum].palletwait=station[0].palletwait;
          station[stanum].busy=false;
          station[stanum].done=false;
          strcpy(station[stanum].robot_time,station[0].robot_time);
          station[stanum].rtp1=station[0].rtp1;
          station[stanum].rtp2=station[0].rtp2;
          station[stanum].rtp3=station[0].rtp3;
          for (i=0; i<10; i++)
            {
              station[stanum].rtp4[i]=station[0].rtp4[i];
              station[stanum].rtp5[i]=station[0].rtp5[i];
            }
        }
      else if (opt==2)
        {
            separate_unloader=true;
            strcpy(station[stanum].name,"remove");
            change_station_conveyor(stanum);
            station[stanum].busy=false;
            station[stanum].done=false;
            prompt_for_robot_distribution(stanum);
        }
    }
}

////////////////////////////////////////////////////// DISPLAY STATION ROBOT

void display_station_robot(int stanum)
{
 int done_user,index,perc;

 printf("<2>  Robot distribution:");

 if ( strcmp(station[stanum].robot_time,"cons")==0)
     printf("\tConstant (%d)\n\n",                    station[stanum].rtp1);

 else if ( strcmp(station[stanum].robot_time,"tria")==0)
   {  printf("\tTriangular (%d, %d, %d)\n\n",      station[stanum].rtp1,
                                                    station[stanum].rtp2,
                                                    station[stanum].rtp3);
   }
 else if ( strcmp(station[stanum].robot_time,"norm")==0)
   {  printf("\tNormal (%d, %d)\n\n",          station[stanum].rtp1,
                                                    station[stanum].rtp2);
   }
 else if ( strcmp(station[stanum].robot_time,"unif")==0)
   {  printf("\tUniform (%d, %d)\n\n",             station[stanum].rtp1,
                                                    station[stanum].rtp2);
   }
 else if ( strcmp(station[stanum].robot_time,"expo")==0)
   {  printf("t\tExponential (%d)\n\n",             station[stanum].rtp1);
   }
 else if ( strcmp(station[stanum].robot_time,"user")==0)
   { printf("\tUser defined\n\t\t\ttime:\t\t percentage:\n");
     done_user=false;
     index=0;
     perc=0;
```

```
    while ( !done_user )
      { printf("\t\t\t%d\t\t%d\n",station[stanum].rtp4[index],
                                   station[stanum].rtp5[index]);
        index++;
        perc+=station[stanum].rtp5[index];
        if (station[stanum].rtp5[index]==0 || perc>=100)
           { done_user=true;
             printf("\n");
           }
      }
   }
 else printf("\t\t\tDistribution not defined\n\n");
 // if ( strcmp(waste  ,"1")==0 ) garb_can+=1;
}
//////////////////////////////////////////////////// DISPLAY STATION SERVICE

void display_station_service(int stanum)
{
 int done_user,index,perc;

 printf("<3>  Service distribution:");
 if ( strcmp(station[stanum].service_time,"cons")==0)
     printf("\tConstant (%d)\n\n",                    station[stanum].stp1);

 else if ( strcmp(station[stanum].service_time,"tria")==0)
  {  printf("\tTriangular (%d, %d, %d)\n\n",      station[stanum].stp1,
                                                   station[stanum].stp2,
                                                   station[stanum].stp3);
   }
 else if ( strcmp(station[stanum].service_time,"norm")==0)
  {  printf("\tNormal (%d, %d)\n\n",          station[stanum].stp1,
                                                   station[stanum].stp2);
   }
 else if ( strcmp(station[stanum].service_time,"unif")==0)
  {  printf("\tUniform (%d, %d)\n\n",             station[stanum].stp1,
                                                   station[stanum].stp2);
   }
 else if ( strcmp(station[stanum].service_time,"expo")==0)
  {  printf("\tExponential (%d)\n\n",             station[stanum].stp1);
   }
 else if ( strcmp(station[stanum].service_time,"user")==0)
  { printf("\tUser defined\n\t\t\ttime:\t\t percentage:\n");
    done_user=false;
    index=0;
    perc=0;
    while ( !done_user )
      { printf("\t\t\t%d\t\t%d\n",station[stanum].stp4[index],
                                   station[stanum].stp5[index]);
        index++;
        perc+=station[stanum].stp5[index];
        if (station[stanum].stp5[index]==0 || perc>=100)
           { done_user=true;
             printf("\n");
           }
      }
   }
 else printf("\t\t\tdistribution not defined\n\n");
}

//////////////////////////////////////////////////////// MAIN MENU SIMULAT
```

```
int main_menu_simulat(void)
{
 int valid,sat,opt,ret;
 char resp[20];

 if (trace) fprintf(fpt,"MAIN_MENU_SIMULAT\n");

 sat=false;

 while (!sat)
   {
     clrscr();
     printf ("\t\t\tMAIN MENU FOR SIMULATION \n\n");

     display_current_simulation_scr();

     valid=test_validity_of_parts_req();

     if (valid)
       { printf("<7>  previous screen\n<8>  next screen\n");
         printf("\nEnter a line number to edit that line's information \n");
         printf("  or to invoke the desired option: ");
         opt=read_integer();

         while (opt<1 || opt>8)
           {
             printf("\nThe response entered is not valid.  Please choose");
             printf(" from the displayed\n line numbers (line numbers are ");
             printf("in < > brackets): ");
             opt=read_integer();
           }

         if      (opt==1)        change_number_pal();
         else if (opt==2)        change_order();
         else if (opt==3)        change_machining_seq();
         else if (opt==4)        change_number_reps();
         else if (opt==5)        change_length_sim();
         else if (opt==6)        change_warm_up();
         else if (opt==7)      {
                                 sat=true;
                                 ret=1;
                               }
         else if (opt==8)      {
                                 sat=true;
                                 ret=3;
                               }

       }
     else
       {
         printf("\nYou MUST change the machining operations required.");
         printf("  Operation(s) are specified\nfor which no corresponding");
         printf(" machining station exists.       press enter: 3");
         gets(resp);

         change_machining_seq();
       }
   }
 return ret;
```

```
}

//////////////////////////////////////////// DISPLAY CURRENT SIMULATION SCR

void display_current_simulation_scr(void)
{
 int j,done;

 if (trace) fprintf(fpt,"display_current_simulation_scr\n");

 //clrscr();
 printf("<1>  Maximum Number of Pallets Allowed (max parts in system):\t%d\n",
                                                    max_pallets_in_system);
 if (inorder)
    printf("<2>  Processing by machines is to be done:     IN THIS ORDER\n\n");
 else
    printf("<2>  Processing by machines is to be done:     IN ANY ORDER\n\n");

 display_current_operations_scr();

 printf("<4>  Number of Replications:\t\t%d\n",reps);
 printf("<5>  Length of each replication:\t%ld seconds\t(%ld hours)\n",
                                                    length_of_sim,
                                                    length_of_sim/3600 );
 printf("<6>  Warm-up period:\t\t\t%ld seconds\t(%ld hours)\n\n",
                                                    warm_up,
                                                    warm_up/3600);
}

//////////////////////////////////////////// DISPLAY CURRENT OPERATIONS SCR

void display_current_operations_scr(void)
{
 int j,done;

 printf("<3>  Machining operations required:");
 printf("\t\t   %s\n",mach_seq_reqd[0]);

 done=false;
 j=1;
 while (!done)
   { if (strcmp(mach_seq_reqd[j],"remove")==0)
       {
         printf("\t\t\t\t\t\t   %s\n\n",mach_seq_reqd[j]);
         done=true;
       }
     else
       {
         printf("\t\t\t\t\t\t   %s\n",mach_seq_reqd[j]);
         j++;
       }
   }

}

//////////////////////////////////////////////////////// CHANGE NUMBER PAL

void change_number_pal(void)
{
 char* waste;
```

```
  char resp[20];

  if (trace) fprintf(fpt,"CHANGE_NUMBER_PALLETS\n");

  printf("\nenter the new Maximum Number of Pallets (1,2, ... ,20): ");
  gets(resp);
  while ((strcmp(resp,"1")!=0) &&(strcmp(resp,"2")!=0) &&
         (strcmp(resp,"3")!=0) &&(strcmp(resp,"4")!=0) &&
         (strcmp(resp,"5")!=0) &&(strcmp(resp,"6")!=0) &&
         (strcmp(resp,"7")!=0) &&(strcmp(resp,"8")!=0) &&
         (strcmp(resp,"9")!=0) &&(strcmp(resp,"10")!=0)&&
         (strcmp(resp,"11")!=0)&&(strcmp(resp,"12")!=0)&&
         (strcmp(resp,"13")!=0)&&(strcmp(resp,"14")!=0)&&
         (strcmp(resp,"15")!=0)&&(strcmp(resp,"16")!=0)&&
         (strcmp(resp,"17")!=0)&&(strcmp(resp,"18")!=0)&&
         (strcmp(resp,"19")!=0)&&(strcmp(resp,"20")!=0) )

       { printf("You must select from (1,2,3, ... ,20), please re-enter: ");
         gets(resp);
       }

  max_pallets_in_system=atoi(resp);

 }

 ////////////////////////////////////////////////// TEST VALIDITY OF PARTS REQ

 int test_validity_of_parts_req(void)
 {
  int i,j,k,l,n,max,ref,sat,notfound;
  int a,b,redo,found;

  char* waste;
  char resp[20];

  if (trace) fprintf(fpt,"TEST_VALIDITY_OF_PARTS_REQ\n");

  redo=false;

  for (a=0; a<=10; a++)
    {
      found=false;

      for (b=0; b<=num_machines+separate_unloader+1; b++)
        {
          if ((strcmp(mach_seq_reqd[a],station[b].name)==0)||
              (strcmp(mach_seq_reqd[a],"remove")==0))
             found=true;
        }

      if (!found)
        {
          printf("%s station does not exist,",mach_seq_reqd[a]);
          printf(" operation must be changed or removed.\n");

          redo=true;
        }
    }

  if (redo) return false;
```

```
 else return true;

}

////////////////////////////////////////////////////// CHANGE MACHINING SEQ

void change_machining_seq(void)
{
 int i,j,k,l,n,max,ref,sat,notfound;
 int a,b,redo,found;

 char* waste;
 char resp[20];

 if (trace) fprintf(fpt,"CHANGE_MACHINING_SEQ\n");

 clrscr();

 max=num_machines;
 clrscr();
 printf("\t\tCHANGE MACHINING SEQUENCE REQUIRED\n\n");

 //display_current_operations_scr();

 printf("There are currently %d machine(s).\nThe number ",max);
 printf("of machining operations cannot exceed %d.\n\n",max);

 printf("How many machining operations does each part");
 printf(" require?\n(enter 1,2,...,%d):",num_machines);
 n=read_integer();

 while (n<1 || n>num_machines)
   { printf("the number of operations must be between 1 and");
     printf(" %d, the number of machines. please re-enter: ",max);
     n=read_integer();
   }

 strcpy(mach_seq_reqd[0],"enter");

 printf("\nEnter the required machine operation by entering ");
 printf("the corresponding integer\n");
 for (i=1;i<=n;i++)
    {
      for (j=1;j<=num_machines;j++)
         printf("\t\t\t<%d>:\t%s\n",j,station[j].name);

      printf("\tmachine operation #%d: ",i);
      ref=read_integer();

      while (ref<1 || ref>num_machines)
        {
           printf("you must select from (1,...,");
           printf("%d), please re-enter: ",max);
           ref=read_integer();
        }

      strcpy(mach_seq_reqd[i],station[ref].name);
    }

 for (k=n+1;k<=10;k++)
```

```
    strcpy(mach_seq_reqd[k],"remove");
}

////////////////////////////////////////////////////////////////// CHANGE_ORDER

void change_order(void)
{
 char* waste;
 char resp[20];

 printf("Machining operations can be required to be done:\n\n<1>  ");
 printf("specifically in the order listed (IN ORDER)\n<2>  in any order");
 printf(" as machines become available (ANY ORDER)\n\n");
 printf("(enter 1= IN ORDER, or 2= ANY ORDER): ");
   gets(resp);

 while ( (strcmp(resp,"1")!=0)&&(strcmp(resp,"2")!=0) )
   {
     printf("you must select from (1,2), please re-enter: ");
       gets(resp);

   }

 if (strcmp(resp,"1")==0)
   inorder=true;
 else
   inorder=false;
}

////////////////////////////////////////////////////////////////// CHANGE NUM REPS

void change_number_reps(void)
{
 char* waste;
 char resp[20];


 if (trace) fprintf(fpt,"CHANGE_NUMBER_REPS\n");

 printf("\nenter the desired Number of Replications (1,2, ... ,20): ");

   gets(resp);
 while ( (strcmp(resp,"1")!=0 )&&(strcmp(resp,"2")!=0 )&&
         (strcmp(resp,"3")!=0 )&&(strcmp(resp,"4")!=0 )&&
         (strcmp(resp,"5")!=0 )&&(strcmp(resp,"6")!=0 )&&
         (strcmp(resp,"7")!=0 )&&(strcmp(resp,"8")!=0 )&&
         (strcmp(resp,"9")!=0 )&&(strcmp(resp,"10")!=0)&&
         (strcmp(resp,"11")!=0)&&(strcmp(resp,"12")!=0)&&
         (strcmp(resp,"13")!=0)&&(strcmp(resp,"14")!=0)&&
         (strcmp(resp,"15")!=0)&&(strcmp(resp,"16")!=0)&&
         (strcmp(resp,"17")!=0)&&(strcmp(resp,"18")!=0)&&
         (strcmp(resp,"19")!=0)&&(strcmp(resp,"20")!=0)   )

   { printf("You must select from (1,2,3, ... ,20), please re-enter: ");
     gets(resp);
   }
 reps=atoi(resp);

}
```

```
////////////////////////////////////////????//////////////////////// ADD A MACHINE

void add_a_machine(void)
{
 char resp[20];
 struct stations temp;
 int which,i,follow;

 if (trace) fprintf(fpt,"ADD A MACHINE\n");
 if (num_machines==9)
   {
     printf("the number of machines cannot exceed 9. press enter\n");
     gets(resp);
   }
 else
   { which=0;
     num_machines++;
     printf("would you like to duplicate an existing machine? enter (y|n): ");
     gets(resp);
     while ((strcmp(resp,"y")!=0)&&(strcmp(resp,"n")!=0))
       {
         printf("please enter either a y or an n: ");
         gets(resp);
       }
     if (strcmp(resp,"y")==0)
       {
         printf("what machine number? enter (1,2,...,%d): ",num_machines-1);
         which=read_integer();

         while (which<1 || which>num_machines-1)
           {
             printf("you must select from (1,2,...,%d), please re-enter: ",
                                                    num_machines-1);
             which=read_integer();
           }
         temp=station[which];
         printf("enter the machine number that the new machine is to follow: ");
         follow=read_integer();
         while (which<0 || which>=num_machines)
           {
             printf("you must select from (0,1,2,...,%d), please re-enter: ",
                                                    num_machines-1);
             which=read_integer();
           }
       }
     else
       {
         printf("enter the machine number that the new machine is to follow: ");
         follow=read_integer();

         while (which<0 || which>=num_machines)
           {
             printf("you must select from (0,1,2,...,%d), please re-enter: ",
                                                    num_machines-1);
             which=read_integer();
           }

         change_station_name(10);
         clrscr();
```

```
          change_station_conveyor(10);
          prompt_for_robot_distribution(10);
          prompt_for_service_distribution(10);
          temp=station[10];
        }


      for (i=10; i>follow+1; i--)
         station[i]=station[i-1];
      station[follow+1]=temp;
    }

}

////////////////////////////////////////////////////////// REMOVE A MACHINE

void remove_a_machine(void)
{
 char resp[20];
 int which,i;

 if (trace) fprintf(fpt,"REMOVE A MACHINE\n");
 if (num_machines==0)
   {
     printf("the number of machines is already 0. press enter\n");
     gets(resp);
   }
 else
   { which=0;
     num_machines--;

     printf("enter the number of the machine to be removed (1,2,...,%d): ",
                                                    num_machines+1);
         which=read_integer();

         while (which<1 || which>num_machines+1)
           {
             printf("you must select from (1,2,...,%d), please re-enter: ",
                                                      num_machines+1);
             which=read_integer();
           }
     for (i=which; i<10; i++)
        station[i]=station[i+1];
   }
}


////////////////////////////////////////////////////////// CHANGE LENGTH SIM

void change_length_sim(void)
{
 long int sec_hr;
 int num;

 if (trace) fprintf(fpt,"CHANGE_LENGTH_SIM\n");

 printf("\nenter the new duration (in hours 1,2, ... ,10): ");
 num=read_integer();

 while (num<1 || num>10)
```

```
    { printf("You must select from (1,2,3, ... ,10), please re-enter: ");
      num=read_integer();
    }
  sec_hr=3600;
  length_of_sim=num*sec_hr;
  fl_length_of_sim=(float)length_of_sim;

 }

 ////////////////////////////////////////////////////////////// CHANGE WARM UP

 void change_warm_up(void)
 {
  int num;
  long int sec_hr;

  if (trace) fprintf(fpt,"CHANGE_WARM_UP\n");

  printf("\nenter the new warm-up period (in hours 1,2, ... ,10): ");
  num=read_integer();

  while (num<1 || num>10)
    {
      printf("You must select from (1,2,3, ... ,10), please re-enter: ");
      num=read_integer();
    }
  sec_hr=3600;
  warm_up=num*sec_hr;

 }

 ////////////////////////////////////////////////////////////// MAIN MENU OUTPUT

 int main_menu_output(void)
 {
  int num,sat,opt,ret,ok,l;
  char resp[20];

  if (trace) fprintf(fpt,"MAIN_MENU_OUTPUT\n");
  sat=false;

  while (!sat)
    {
      clrscr();
      printf ("\t\t\tMAIN MENU FOR OUTPUT SPECIFICATION\n\n");

      if (outfile)
          printf("<1>  Output File:\t%s\n\n",outfile_name);
      else
        {
          printf("<1>  Output File:\tnot specified.\n");
          printf("\n\t\tOutput will be displayed on screen, but NOT saved\n\n");
        }
      if (debug)
        {
          printf("<2>  Dubugger:\t\tON.\n\n");
          printf("<3>  Replications:\t%d\n\n",reps);
        }
      else
        {
```

```
    printf("<2>  Dubugger:\t\tOFF.\n\n");
    printf("<3>  Replications:\t%d\n\n",reps);
  }


printf("<4>  previous screen\n<5>  run the simulation\n\n<6>  save ");
printf("the current configuration to file\n<7>  quit, do NOT run\n\n");
if ( (debug) && (reps > 1) )
  {
    printf("It is advised to only run one replication in debug ");
    printf("mode\n\tsince quite a lot of output is produced.\n");
  }
printf("\nEnter a line number to edit that line's information \n");
printf("  or to invoke the desired option: ");
opt=read_integer();

while (opt<1 ||opt>7)
  {
    printf("\nThe response entered is not valid.  Please choose");
    printf(" from the displayed\n line numbers (line numbers are ");
    printf("in < > brackets): ");
    opt=read_integer();
  }

if (opt==1 && outfile)
  {
    fclose(fp);
    outfile=false;
  }
else if (opt==1)
  {
    ok=false;
    while (!ok)
      {
       printf("enter a name for the output file (max 20 chars): ");
       gets(outfile_name);
       l=strlen(outfile_name);
       if (l>20)
         printf("length of file name must not exceed 20 characters\n");
       else
         {
           fp=fopen(outfile_name,"w");
           outfile=true;
           ok=true;
         }
      }
  }
else if (opt==2)
  {
    if (debug)
      {
            fclose(fpdb);
            debug=false;
      }
    else
      {
            debug=true;
            fpdb=fopen("sim0.dbg","w");
      }
  }
```

```
    else if (opt==3)
      {
        printf("enter new Number of Replications? (1,2, ... ,20): ");
        num=read_integer();
        while ( num<1 || num>20)
          {
            printf("You must select from (1,2,3, ... ,20),");
            printf(" please re-enter: ");
            num=read_integer();
          }

        reps=num;
      }
    else if (opt==4)
      {
        ret=2;
        sat=true;
      }
    else if (opt==5)
      {
        ret=4;
        sat=true;
      }
    else if (opt==6)
      { ok=false;
        while (!ok)
         {
           printf("enter a name for the configuration file (max 20 chars): ");
           gets(confile_name);
           l=strlen(confile_name);
           if (l>20)
            printf("length of file name must not exceed 20 characters\n");
           else
            {
              fpc=fopen(confile_name,"w");
              ok=true;
            }
         }

        write_confile();
        printf("current configuration saved to: %s\n",confile_name);
        printf("press enter: ");
        gets(resp);
        fclose(fpc);
      }

    else
      {
        ret=5;
        sat=true;
      }

  }

 return ret;
}

////////////////////////////////////////////// PROMPT FOR ROBOT DISTRIBUTION

void prompt_for_robot_distribution(int index)
```

```
{

int okay,sat,rtp_index;
int min,max,avg,mean,vari,tim,perc,total,remain;
char* waste;
char resp[20];

 if (trace) fprintf(fpt,"PROMPT_FOR_ROBOT_DISTRRIBUTION\n");

 clrscr();
 if (index<10)
   printf("\tselect a distribution for station %d's robot times:\n\n",index);
 else
   printf("\tselect a distribution for the new station's robot times:\n\n");

 printf("\t\t\t1: constant\n\t\t\t2: triangular\n\t\t\t3: normal\n\t\t\t4:");
 printf(" uniform\n\t\t\t5: exponential\n\t\t\t6: user defined\n\n");
 printf("\tenter from (1,...,6): ");
   gets(resp);
 while ( (strcmp(resp,"1")!=0) && (strcmp(resp,"2")!=0) &&
         (strcmp(resp,"3")!=0) && (strcmp(resp,"4")!=0) &&
         (strcmp(resp,"5")!=0) && (strcmp(resp,"6")!=0) )
   { printf("you must select from (1,2,3,4,5,6), please re-enter: ");
      gets(resp);
   }
 if (strcmp(resp,"1")==0)
       { strcpy(station[index].robot_time,"cons");
         printf("\n\t\t\tConstant distribution:\n");

         printf("\t\t\tload/unload time (in seconds): ");
         station[index].rtp1=read_integer();
       }
 else if (strcmp(resp,"2")==0)
       { strcpy(station[index].robot_time,"tria");
         okay=false;
         while (!okay)
          {
            printf("\n\t\t\tTriangular distribution:\n");.

            printf("\t\t\tminimum time (in seconds): ");
            min=read_integer();
            station[index].rtp1=min;

            printf("\t\t\taverage time (in seconds): ");
            avg=read_integer();
            station[index].rtp2=avg;

            printf("\t\t\tmaximum time (in seconds): ");
            max=read_integer();
            station[index].rtp3=max;

            if ( (min<=avg) && (avg<=max) )
              okay=true;
            else
              { printf("The values entered do not make sense.\n");
                printf("values must conform to min <= avg <= max.\n");
                printf("\n please re-enter.\n");
              }
          }
       }
```

```
  else if (strcmp(resp,"3")==0)
        { strcpy(station[index].robot_time,"norm");
          printf("\n\t\t\tNormal distribution:\n");

          printf("\t\t\tmean time (in seconds): ");
          mean=read_integer();
          station[index].rtp1=mean;

          printf("\t\t\tvariance (in seconds): ");
          vari=read_integer();
          station[index].rtp2=vari;

        }
  else if (strcmp(resp,"4")==0)
        { strcpy(station[index].robot_time,"unif");
          okay=false;
          while (!okay)
           {
             printf("\n\t\t\tUniform distribution:\n");

             printf("\t\t\tminimum time (in seconds): ");
             min=read_integer();
             station[index].rtp1=min;

             printf("\t\t\tmaximum time (in seconds): ");
             max=read_integer();
             station[index].rtp2=max;

             if (min<=max)
               okay=true;
             else
               { printf("The values entered do not make sense.\n");
                 printf("values must conform to min <= max.\n");
                 printf("\n please re-enter.\n");
               }
           }
        }
  else if (strcmp(resp,"5")==0)
        { strcpy(station[index].robot_time,"expo");
          printf("\n\t\t\tExponential distribution:\n");

          printf("\t\t\tmean time (in seconds): ");
          min=read_integer();
          station[index].rtp1=min;
        }
  else if (strcmp(resp,"6")==0)
        { strcpy(station[index].robot_time,"user");
          okay=false;
          while (!okay)
            { printf("\n\t\t\tUser defined distribution:\n\n\t\tenter times ");
              printf("and their associated\n");
              printf("\t\tpercentages to define a distribution.\n\n");

              total=0;
              rtp_index=0;
              while (total < 100)
                { printf("\t\tenter a time (in seconds): ");
                  tim=read_integer();
                  station[index].rtp4[rtp_index]=tim;
```

```
                printf("\t\tenter a percentage (10,12,50,...): \n");

                printf("\t\t       %d remaining....          : ",100-total);
                perc=read_integer();
                station[index].rtp5[rtp_index]=perc;
                total += perc;

                rtp_index+=1;
                if (rtp_index > 10)
                  { printf("Error: too many times....\n");
                    total=200;
                  }
              }

            if ( total == 100 )
             okay=true;
            else
             { printf("The values entered do not make sense.\n");
               printf("Percentages must total 100, and a maximum of 10\n");
               printf("   times are permitted.   please re-enter.\n");
             }
          }
      }

}

//////////////////////////////////////////// PROMPT FOR SERVICE DISTRIBUTION

void prompt_for_service_distribution(int index)
{

int okay,sat,stp_index;
int min,max,avg,mean,vari,tim,perc,total;
char resp[20];

 if (trace) fprintf(fpt,"PROMPT_FOR_SERVICE_DISTRRIBUTION\n");

 clrscr();
 if (index<10)
   printf("\tselect a distribution for station %d's service time:\n\n",index);
 else
   printf("\tselect a distribution for the new station's service times:\n\n");

 printf("\t\t\t1: constant\n\t\t\t2: triangular\n\t\t\t3: normal\n\t\t\t4: ");
 printf("uniform\n\t\t\t5: exponential\n\t\t\t6: user defined\n\n");
 printf("\tenter from (1,...,6): ");
   gets(resp);
 while ( (strcmp(resp,"1")!=0) && (strcmp(resp,"2")!=0) &&
         (strcmp(resp,"3")!=0) && (strcmp(resp,"4")!=0) &&
         (strcmp(resp,"5")!=0) && (strcmp(resp,"6")!=0) )
       { printf("you must select from (1,2,3,4,5,6), please re-enter: ");
         gets(resp);
       }
 if (strcmp(resp,"1")==0)
       { strcpy(station[index].service_time,"cons");
         printf("\n\t\t\tConstant distribution:\n");
         printf("\t\t\tservice time (in seconds): ");
         station[index].stp1=read_integer();
       }
 else if (strcmp(resp,"2")==0)
```

```
      { strcpy(station[index].service_time,"tria");
        okay=false;
        while (!okay)
         {
           printf("\n\t\t\tTriangular distribution:\n");

           printf("\t\t\tminimum time (in seconds): ");
           min=read_integer();
           station[index].stp1=min;

           printf("\t\t\taverage time (in seconds): ");
           avg=read_integer();
           station[index].stp2=avg;

           printf("\t\t\tmaximum time (in seconds): ");
           max=read_integer();
           station[index].stp3=max;

           if ( (min<=avg) && (avg<=max) )
             okay=true;
           else
             { printf("The values entered do not make sense.\n");
               printf("values must conform to min <= avg <= max.\n");
               printf("\n please re-enter.\n");
             }
         }
      }
else if (strcmp(resp,"3")==0)
      { strcpy(station[index].service_time,"norm");
        printf("\n\t\t\tNormal distribution:\n");

        printf("\t\t\tmean time (in seconds): ");
        mean=read_integer();
        station[index].stp1=mean;

        printf("\t\t\tvariance (in seconds): ");
        vari=read_integer();
        station[index].stp2=vari;

      }
else if (strcmp(resp,"4")==0)
      { strcpy(station[index].service_time,"unif");
        okay=false;
        while (!okay)
         {
           printf("\n\t\t\tUniform distribution:\n");

           printf("\t\t\tminimum time (in seconds): ");
           min=read_integer();
           station[index].stp1=min;

           printf("\t\t\tmaximum time (in seconds): ");
           max=read_integer();
           station[index].stp2=max;

           if (min<=max)
             okay=true;
           else
             { printf("The values entered do not make sense.\n");
               printf("values must conform to min <= max.\n");
```

```
                printf("\n please re-enter.\n");
              }
          }
      }
 else if (strcmp(resp,"5")==0)
      { strcpy(station[index].service_time,"expo");
        printf("\n\t\t\tExponential distribution:\n");

        printf("\t\t\tmean time (in seconds): ");
        min=read_integer();
        station[index].stp1=min;
      }
 else if (strcmp(resp,"6")==0)
      { strcpy(station[index].service_time,"user");
        okay=false;
        while (!okay)
          { printf("\n\t\t\tUser defined distribution:\n\n\t\tenter times ");
            printf("and their associated\n");
            printf("\t\tpercentages to define a distribution.\n\n");

            total=0;
            stp_index=0;
            while (total < 100)
              { printf("\t\tenter a time (in seconds): ");
                tim=read_integer();
                station[index].stp4[stp_index]=tim;

                printf("\t\tenter a percentage (10,12,50,...): \n");
                printf("\t\t       %d remaining....           : ",100-total);
                perc=read_integer();
                station[index].stp5[stp_index]=perc;
                total += perc;

                stp_index++;
                if (stp_index > 10)
                  { printf("Error: too many times....\n");
                    total=200;
                  }
              }

            if ( total == 100 )
             okay=true;
            else
             { printf("The values entered do not make sense.\n");
               printf("Percentages must total 100, and a maximum of 10\n");
               printf("   times are permitted.   please re-enter.\n");
             }
          }
      }

}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                        SIMULATION  MODULES                         //
//                                                                    //
////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////// CREATE_EVENT

event* create_event (void)
{
 event* temp;

 if (trace) fprintf(fpt,"CREATE_EVENT\n");

 temp = new event;
 strcpy(temp->name,"end");
 temp->time=0;
 temp->partnum=0;
 temp->isempty=true;
 temp->inorder=inorder;
 temp->next=0;
 return temp;
}

//////////////////////////////////////////////////////////////START_ONE_THRU

void start_one_thru(long int time)
{
 int i;

 if (trace) fprintf(fpt,"START_ONE_THRU\n");

 current=create_event();
 current->isempty=false;
 current->partnum=part_num;
 part_num++;
 strcpy(current->name,"entersys");
 current->stanum=0;
 current->time=time;
 current->artime=time;
 current->totconvtime=0;
 current->inorder=inorder;
 for (i=0; i<11; i++)
   strcpy(current->need[i],mach_seq_reqd[i]);

 current->next=0;
 parts_in_system+=1;
 //pallets_in_system++;
 schedule_current();
}

//////////////////////////////////////////////////////////////// SCHEDULE_CURRENT

void schedule_current(void)
{
 event* temp;
 event* trail;

 if (trace) fprintf(fpt,"SCHEDULE_CURRENT\n");

 if ( strcmp(current->name,"arrive")==0)
```

```
    // it is going into calender1 (conveyor)

  { if (frozen)

       current->time = freeze_till_time+interstation_time;

    if (strcmp(calender1->name,"end")==0)
           // calender1 is empty, add event to beginning
     { current->next=calender1;
       calender1=current;
     }
    else if (current->time<calender1->time)
                // calender1 is not empty, add event to beginning
     { current->next=calender1;
       calender1=current;
     }
    else  // find appropriate place and insert event
     { trail=calender1;
       temp=calender1->next;
       while ((strcmp(temp->name,"end") != 0) &&
              (temp->time <= current->time))
         { trail=temp;
           temp=temp->next;
         }
       current->next=temp;
       trail->next=current;
     }
    current->stime=tnow;
  }
 /*
 else if ( (strcmp(current->name,"unbusysta")==0) ||
           (strcmp(current->name,"leavesta" )==0) ||
           (strcmp(current->name,"leaveserv")==0) )

  { // event will be inserted into calender2 and should
    // precede all events with same time.

    if (strcmp(calender2->name,"end")==0)
       // cal2 is empty, add event to beginning
     { current->next=calender2;
       calender2=current;
     }
    else if (current->time <= calender2->time)
            // cal2 is not empty, add record to beginning
     { current->next=calender2;
       calender2=current;
     }
    else  // find appropriate place and insert event
     { trail=calender2;
       temp=calender2->next;
       while ((strcmp(temp->name,"end") != 0) &&
              (temp->time < current->time))
         { trail=temp;
           temp=temp->next;
         }
       current->next=temp;
       trail->next=current;
     }
  }*/
 else  // event will be inserted into calender2
```

```
   {
    if (strcmp(calender2->name,"end")==0)
              // cal2 is empty, add event to beginning
     { current->next=calender2;
       calender2=current;
     }
    else if (current->time<calender2->time)
            // cal2 is not empty, add record to beginning
     { current->next=calender2;
       calender2=current;
     }
    else  // find appropriate place and insert event
     { trail=calender2;
       temp=calender2->next;
       while ((strcmp(temp->name,"end") != 0) &&
              (temp->time < current->time))              //was <=
         { trail=temp;
           temp=temp->next;
         }
       current->next=temp;
       trail->next=current;
     }
    current->stime=tnow;
   }
 }

 ////////////////////////////////////////////////////////////// GET NEXT TIME

 long int get_next_time(void)
 {
   //event*temp
  long int ans;

  if (trace) fprintf(fpt,"GET_NEXT_TIME\n");

  if ((strcmp(calender1->name,"end")==0) &&
      (strcmp(calender2->name,"end")==0))

    { if (trace)
      fprintf(fpt,"ATTEMPTED REMOVAL OF NEXT TIME WITH BOTH LISTS EMPTY!\n");
      ans=-1;

    }

   else if (strcmp(calender1->name,"end")==0)
    { //temp=calender2;
      ans=calender2->time;

    }
   else if (strcmp(calender2->name,"end")==0)
    { //temp=calender1;
      ans=calender1->time;

    }
  /* else if (calender1->time == calender2->time)

         { if (calender1->stime < calender2->stime)
              ans=calender1->time;

           else if (calender1->stime >= calender2->stime)
```

```
            ans=calender2->time;

        }
    */
   else if (calender1->time <= calender2->time)
     { //temp=calender1;
       ans=calender1->time;

     }
   else if (calender1->time > calender2->time)
     { //temp=calender2;
       ans=calender2->time;

     }
   else
     { if (trace)
       fprintf(fpt,"GET_NEXT TIME !!!!THIS CANNOT HAPPEN!!!!\n");
       ans=-1;

     }
 return ans;
}

////////////////////////////////////////////////////////////////// REMOVE_NEXT

event* remove_next(void)
{
 event* temp;

 if (trace) fprintf(fpt,"REMOVE_NEXT\n");

 if ((strcmp(calender1->name,"end")==0) &&
      (strcmp(calender2->name,"end")==0))
    { if (trace)
      fprintf(fpt,"ATTEMPTED REMOVAL OF EVENT WITH BOTH LISTS EMPTY!\n");
      temp=calender1;

    }
  else if (strcmp(calender1->name,"end")==0)
    { temp=calender2;
      calender2=calender2->next;

    }
  else if (strcmp(calender2->name,"end")==0)
    { temp=calender1;
      calender1=calender1->next;

    }
  /*else if (calender1->time == calender2->time)
    { temp=calender2;
      calender2=calender2->next;

    }*/
  else if (calender1->time <= calender2->time)
    { temp=calender1;
      calender1=calender1->next;

    }
  else if (calender1->time > calender2->time)
```

```
    { temp=calender2;
      calender2=calender2->next;

    }
  else
    { if (trace)
      fprintf(fpt,"REN_NEXT MOST PECULIAR BABY, ROLL\n");
      temp=calender1;

    }
  return temp;
}

////////////////////////////////////////////////////////DETERMINE_ROBOT_TIME

long int determine_robot_time(int stanum)
{
 long int time;
 double x,sum,bp,cap;
 int i,j,diff,bma,cma,cmb,index,done;

 if (trace) fprintf(fpt,"DETERMINE_ROBOT_TIME\n");


 if      ( strcmp(station[stanum].robot_time,"cons")==0 )
     time=station[stanum].rtp1;

 else if ( strcmp(station[stanum].robot_time,"unif")==0 )
    {
      x=rand()/(float)RAND_MAX;
      diff=station[stanum].rtp2-station[stanum].rtp1+1;
      time=(station[stanum].rtp1+x*diff);
    }
 else if ( strcmp(station[stanum].robot_time,"expo")==0 )
    {
      x=rand()/(float)RAND_MAX;
      time=-1*log(x)*station[stanum].rtp1;
    }
 else if ( strcmp(station[stanum].robot_time,"norm")==0 )
    {
      sum=0.0;
      for (i=1; i<=12; i++)
        { x=rand()/(float)RAND_MAX;
          sum+=x;
        }
      time=station[stanum].rtp1+(sum-6)*station[stanum].rtp2;
    }
 else if ( strcmp(station[stanum].robot_time,"tria")==0 )
    {
      x=rand()/(float)RAND_MAX;

      bma=station[stanum].rtp2-station[stanum].rtp1;
      cma=station[stanum].rtp3-station[stanum].rtp1;
      cmb=station[stanum].rtp3-station[stanum].rtp2;

      bp= bma/ (float)cma;

      if (x<bp)
        time=station[stanum].rtp1+sqrt(bma*cma*x);
      else
```

```
        time=station[stanum].rtp3-sqrt(cmb*cma*(1-x));
    }
  else
    {
      x=100*(rand()/(float)RAND_MAX);
      index=0;
      cap=0.0;
      done=false;
      while (!done)
        { cap+=station[stanum].rtp5[index];
          if (x<=cap)
            {
              time=station[stanum].rtp4[index];
              done=true;
            }
          index++;
        }
    }

  return time;
}

////////////////////////////////////////////////////DETERMINE_SERVICE_TIME

long int determine_service_time(int stanum)
{
  long int time;
  double x,sum,bp,cap;
  int i,j,diff,bma,cma,cmb,index,done;

  if (trace) fprintf(fpt,"DETERMINE_SERVICE_TIME\n");


  if       ( strcmp(station[stanum].service_time,"cons")==0 )
      time=station[stanum].stp1;

  else if ( strcmp(station[stanum].service_time,"unif")==0 )
    {
      x=rand()/(float)RAND_MAX;
      diff=station[stanum].stp2-station[stanum].stp1+1;
      time=(station[stanum].stp1+x*diff);
    }
  else if ( strcmp(station[stanum].service_time,"expo")==0 )
    {
      x=rand()/(float)RAND_MAX;
      time=-1*log(x)*station[stanum].stp1;
    }
  else if ( strcmp(station[stanum].service_time,"norm")==0 )
    {
      sum=0.0;
      for (i=1; i<=12; i++)
        { x=rand()/(float)RAND_MAX;
          sum+=x;
        }
      time=station[stanum].stp1+(sum-6)*station[stanum].stp2;
    }
  else if ( strcmp(station[stanum].service_time,"tria")==0 )
    {
      x=rand()/(float)RAND_MAX;
```

```
      bma=station[stanum].stp2-station[stanum].stp1;
      cma=station[stanum].stp3-station[stanum].stp1;
      cmb=station[stanum].stp3-station[stanum].stp2;

      bp= bma/ (float)cma;

      if (x<bp)
        time=station[stanum].stp1+sqrt(bma*cma*x);
      else
        time=station[stanum].stp3-sqrt(cmb*cma*(1-x));
    }
  else
    {
      x=100*(rand()/(float)RAND_MAX);
      index=0;
      cap=0.0;
      done=false;
      while (!done)
        { cap+=station[stanum].stp5[index];
          if (x<=cap)
            {
              time=station[stanum].stp4[index];
              done=true;
            }
          index++;
        }
    }

  return time;
}


////////////////////////////////////////////////////// DELAY_PARTS_ON_CONVEYOR

 void delay_parts_on_conveyor(long int time)
 {
   event* temp;
   long int t;

   if (trace) fprintf(fpt,"DELAY_PARTS_ON_CONVEYOR\n");
   if (frozen)
     { t=tnow + time - freeze_till_time;
       if (t>0)
         { temp=calender1;
           while( strcmp(temp->name,"end") != 0)
              { if (temp->time != tnow)
                   temp->time+=t;

                temp=temp->next;
              }
           freeze_till_time += t;

         }
     }
   else
     { t=time;
       temp=calender1;
       while( strcmp(temp->name,"end") != 0)
          { if (temp->time != tnow)
                 temp->time+=t;
```

```
            temp=temp->next;
          }
        frozen=true;
        freeze_till_time=tnow+t;

      }
   if (trace) fprintf(fpt,"\t\t\tFREEZE_TILL_TIME: %ld\n",freeze_till_time);
}

////////////////////////////////////////////////////////////////// INC_STANUM

void inc_stanum(void)
{
   if (trace) fprintf(fpt,"INC_STANUM\n");
   current->stanum += 1;
   if (current->stanum > num_machines+separate_unloader)
      current->stanum=0;
}

////////////////////////////////////////////////////////////////// CAN_SERVICE

int can_service(int stanum)
{
 int i, count, spot;

  if (trace) fprintf(fpt,"CAN_SERVICE\n");
  if (!station[stanum].busy)
    {
     if ( current->inorder)
      { if (strcmp(current->need[0],station[stanum].name)==0)
         { count=0;
           while (strcmp(current->need[count],"remove") != 0)
                count++;
           for (i=0;i<count;i++)
              strcpy(current->need[i],current->need[i+1]);
              return true;
         }
        else
             return false;
      }
     else
       { count=0;
         spot=-1;
         while (strcmp(current->need[count],"remove") != 0)
            { if (strcmp(current->need[count],station[stanum].name) == 0)
                 spot=count;
              count++;
            }
          if (spot >= 0)
           { for (i=spot; i<count; i++)
                strcpy(current->need[i], current->need[i+1]);
             return true;
           }
          else
             return false;
       }
    }
  else return false;
 }
```

```
//////////////////////////////////////////////////////////////// ENTER_SYSTEM

void enter_system(void)
{
 int i,stanum,partnum;
 long int time; //,time;

 if (trace) fprintf(fpt,"ENTER_SYSTEM\n");

 if (pallets_in_system <= max_pallets_in_system)
   {
     if (!station[0].busy)
       {
         station[0].busy=true;
         stanum=current->stanum;

         if (station[0].conveyorwait)
           {
             time=determine_robot_time(stanum);
             delay_parts_on_conveyor(time);
             for (i=0; i<10; i++)
               strcpy(current->need[i],current->need[i+1]);

             strcpy(current->name,"leavesta");
             current->time=tnow+time;
             schedule_current();
           }
         else if (station[0].palletwait)
           {
             time=determine_robot_time(stanum);
             for (i=0; i<10; i++)
               strcpy(current->need[i],current->need[i+1]);

             strcpy(current->name,"leaveserv");
             current->time=tnow+time;
             schedule_current();
           }
         else  // station[0] is pallet continues
           {
             time=determine_robot_time(stanum);
             delay_parts_on_conveyor(time);
             for (i=0; i<10; i++)
               strcpy(current->need[i],current->need[i+1]);

             strcpy(current->name,"leavesta");
             current->time=tnow+time;
             schedule_current();
           }
       }
      else
        {
            current->time += 1;
            schedule_current();
        }
   }
  else
    { if (trace)
      fprintf(fpt,"ENTER_SYS TOO MANY PARTS ATTEMPTING ENTRY!\n");
      schedule_current();
    }
```

```
}

//////////////////////////////////////////////////////////// ARRIVE_AT_STATION

void arrive_at_station(void)
{
  long int time,
           time2,
           time3;

  int   stanum,
        partnum,
        i;

  event* temp;

  if (trace) fprintf(fpt,"ARRIVE_AT_STATION\n");

  stanum=current->stanum;
  partnum=current->partnum;

  if ((pallets_in_system + delayed_entry < max_pallets_in_system)&&
                (stanum==1))
   {
       temp=current;
       start_one_thru(tnow);
       pallets_in_system++;
       current=temp;
   }

  if ( (current->isempty) && (current->stanum==0) && (delayed_entry >= 1) )
    {
      delete current;
      start_one_thru(tnow);
      delayed_entry--;
    }

  else if (current->isempty)
    {
      if (station[stanum].busy && station[stanum].done )
      // empty pallet arives at station waiting for an empty pallet
       {
         time=determine_robot_time(stanum);
         delay_parts_on_conveyor(time);
         station[stanum].done=false;

         delete current;

         current=temp_storage[stanum];
         current->isempty=false;
         strcpy(current->name,"leavesta");
         current->time=tnow+time;
         schedule_current();
       }
      else
       {
         inc_stanum();
         current->time=tnow+interstation_time;
         schedule_current();
       }
```

```
  }
else if (station[current->stanum].busy)
  {
     inc_stanum();
     current->time=tnow+interstation_time;
     schedule_current();
  }
else if ( strcmp(current->need[0],"remove") == 0)

  // a completed part is arriving at a station

  {
     if (( current->stanum == 0 && !separate_unloader) ||
         ( current->stanum == num_machines+1 && separate_unloader) )
         {
           time2=0;
           time3=0;

           current->totconvtime += (current->time-current->arconv);
           stanum=current->stanum;
           station[stanum].busy=true;

           if (station[current->stanum].conveyorwait)
             {
               time=determine_robot_time(stanum);
               delay_parts_on_conveyor(time);

               current->time += time;
               update_stats();

               if (stanum == 0 && !separate_unloader )
                 {
                   time2=determine_robot_time(stanum);
                   current->isempty=false;
                   current->partnum=part_num;
                   part_num++;
                   strcpy(current->name,"leavesta");
                   current->stanum=0;
                   current->time=tnow+time+time2;
                   current->artime=tnow+time;
                   current->totconvtime=0;
                   current->inorder=inorder;
                   for (i=0; i<10; i++)
                   strcpy(current->need[i],mach_seq_reqd[i+1]);
                   current->next=0;
                   schedule_current();
                 }
               else
                 { strcpy(current->name,"emptygo");
                   current->time=tnow+time;
                   current->isempty=true;
                   current->stanum=stanum;
                   schedule_current;
                   delayed_entry++;
                 }
             }
           else if (station[stanum].palletwait)
             {
                time=pallet_unload_time;
                delay_parts_on_conveyor(time);
```

```
        time2=determine_robot_time(stanum);

        current->time += (time+time2);
        update_stats();

        if (stanum == 0 && !separate_unloader )
          {
            time3=determine_robot_time(stanum);
            current->isempty=false;
            current->partnum=part_num;
            part_num++;
            strcpy(current->name,"leaveserv");
            current->stanum=0;
            current->time=tnow+time+time2+time3;
            current->artime=tnow+time+time2;
            current->totconvtime=0;
            current->inorder=inorder;
            for (i=0; i<10; i++)
            strcpy(current->need[i],mach_seq_reqd[i+1]);
            current->next=0;
            schedule_current();

          }
        else
          { strcpy(current->name,"leaveserv");
            current->time=tnow+time+time2;
            current->isempty=true;
            current->stanum=stanum;
            schedule_current;
            delayed_entry++;
          }

      }
    else // pallet continues
      {
        time=determine_robot_time(stanum);
        delay_parts_on_conveyor(time);

        current->time += time;
        update_stats();

        if (stanum == 0 && !separate_unloader )
          {
            time2=determine_robot_time(stanum);

            current->isempty=false;
            current->partnum=part_num;
            part_num++;
            strcpy(current->name,"leavesta");
            current->stanum=0;
            current->time=tnow+time+time2;
            current->artime=tnow+time;
            current->totconvtime=0;
            current->inorder=inorder;
            for (i=0; i<10; i++)
            strcpy(current->need[i],mach_seq_reqd[i+1]);
            current->next=0;
            schedule_current();
          }
        else
```

```
                    { strcpy(current->name,"emptygo");
                      current->time=tnow+time;
                      current->isempty=true;
                      current->stanum=stanum;
                      schedule_current;
                      delayed_entry++;
                    }
                }
            }
        else      // arriving at a station it does not need
            {
              inc_stanum();
              current->time += interstation_time;
              schedule_current();
            }
    }
  else if ( can_service(stanum) )
    {
        current->totconvtime += (current->time-current->arconv);

        if (station[stanum].conveyorwait)
          {
            time=determine_robot_time(stanum) +
                 determine_service_time(stanum) +
                 determine_robot_time(stanum);

            delay_parts_on_conveyor(time);
            station[stanum].busy=true;
            current->time=tnow+time;

            strcpy(current->name,"leavesta");
            schedule_current();
          }
        else if (station[current->stanum].palletwait)
          { //current->totconvtime += (current->time - current->arconv);
            //stanum=current->stanum;
            time=pallet_unload_time;
            delay_parts_on_conveyor(time);
            station[stanum].busy=true;
            time2= determine_robot_time(stanum) +
                   determine_service_time(stanum) +
                   determine_robot_time(stanum);

            strcpy(current->name,"leaveserv");
            current->time=tnow+time+time2;
            schedule_current();
          }
        else // not conveyor_wait or pallet_wait
          { //current->totconvtime+=(tnow-current->arconv);
            time=determine_robot_time(stanum);
            delay_parts_on_conveyor(time);

            temp_storage[stanum]=current;

            station[stanum].busy=true;
            station[stanum].done=false;

            current=create_event();
            strcpy(current->name,"donesta");
            time2=determine_service_time(stanum);
```

```
            current->time=tnow+time+time2;
            current->stanum=stanum;
            current->partnum=partnum;
            schedule_current();

            current=create_event();
            strcpy(current->name,"emptygo");
            current->isempty=true;
            current->partnum= 0;
            current->time=tnow+time;
            current->stanum=stanum;
            schedule_current();

          }
    }
  else // not can_service
    {
      current->time=tnow+interstation_time;
      inc_stanum();
      schedule_current();
    }
}

////////////////////////////////////////////////////////////// UNBUSY_STATION

void unbusy_station(void)
 {
   if (trace) fprintf(fpt,"UNBUSY_STATION\n");

   station[current->stanum].busy=false;
   delete current;
}

////////////////////////////////////////////////////////////// LEAVE_STATION

void leave_station(void)
{
  long int time;
  int stanum;

  if (trace) fprintf(fpt,"leave_sta\n");

  station[current->stanum].busy=false;
  station[current->stanum].done=false;

  strcpy(current->name,"arrive");
  current->time=tnow+interstation_time;
  current->arconv=tnow;

  inc_stanum();
  schedule_current();
}

////////////////////////////////////////////////////////////// EMPTYGO STATION

void emptygo_station(void)
{
  long int time;
  int stanum;
```

```
  if (trace) fprintf(fpt,"emptygo_sta\n");

  strcpy(current->name,"arrive");

  current->time=tnow+interstation_time;
  current->arconv=tnow;
  current->isempty=true;

  inc_stanum();
  schedule_current();
}

//////////////////////////////////////////////////////////// LEAVE_SERVICE_PW

void leave_service_pw(void)
{
  long int time;

  if (trace) fprintf(fpt,"LEAVE_SERVICE_PW\n");

  time=pallet_unload_time;
  delay_parts_on_conveyor(time);

  current->time=tnow+time;
  strcpy(current->name,"leaves2");

  schedule_current();
}

//////////////////////////////////////////////////////////////////// LEAVE_S2

void leave_s2(void)
{
  if (trace) fprintf(fpt,"LEAVE_S2\n");

  station[current->stanum].busy=false;

  strcpy(current->name,"arrive");
  current->time=tnow+interstation_time;
  current->arconv=tnow;

  inc_stanum();
  schedule_current();
}

//////////////////////////////////////////////////////////// DONE_STATION

void done_station(void)
{
   if (trace) fprintf(fpt,"FINISH_STATION\n");

   station[current->stanum].done=true;
   delete current;
}




////////////////////////////////////////////////////////////////////////////
```

```
//                                                                          //
//                          STATISTICAL MODULES                             //
//                                                                          //
//////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////// RESET STATS

void reset_stats(void)
{
 int i;

 if (trace) fprintf(fpt,"RESET_REP_STATS\n");

 parts_com=0;
 time_in_syst =0;
 min_t_i_s    =14000;
 max_t_i_s    =0;
 total_parts  =0;
 total_time   =0;
 time_on_conveyor =0;
 min_t_o_c    =14000;
 max_t_o_c    =0;
 total_t_o_c  =0;

 total_time_sq =0;
 total_t_o_c_sq =0;

}

/////////////////////////////////////////////////////////////// UPDATE_STATS

void update_stats(void) // DOES NOT DELETE ENTITY OR ALTER ATTRIBUTES
{
  if (trace) fprintf(fpt,"UPDATE_STATS\n");

  parts_com++;
  time_in_syst=current->time-current->artime;
  if (time_in_syst > max_t_i_s)
     max_t_i_s=time_in_syst;
  if (time_in_syst < min_t_i_s)
     min_t_i_s=time_in_syst;                                  //look into this
  total_parts ++;

  total_time += time_in_syst;
  avg_time_syst=total_time/(float)total_parts;
  total_time_sq += time_in_syst*time_in_syst;
    //current->totconvtime += (tnow-current->arconv);
  time_on_conveyor=current->totconvtime;
  if (time_on_conveyor > max_t_o_c)
     max_t_o_c = time_on_conveyor;
  if (time_on_conveyor < min_t_o_c)
     min_t_o_c = time_on_conveyor;
  total_t_o_c += time_on_conveyor;
  avg_t_o_c=(float)total_t_o_c/total_parts;
  total_t_o_c_sq += time_on_conveyor*time_on_conveyor ;


  /*  fprintf(fp,"AVG:%ld TIS:%ld MiTIS:%ld MaTIS:%ld TPs:%ld TT:%ld TTS:%ld \n"
```

```
        avg_time_syst,
        time_in_syst,
        min_t_i_s,
        max_t_i_s,
        total_parts,
        total_time,
        total_time_sq);
    fprintf(fp,"ATOC:%ld TOC:%ld MiTOC:%ld MaTOC:%ld TTOC:%ld TTOCS:%ld \n",
        avg_t_o_c,
        time_on_conveyor,
        min_t_o_c,
        max_t_o_c,
        total_t_o_c,
        total_t_o_c_sq);    */

}

////////////////////////////////////////////////////////////// RESET MR STATS

void reset_mr_stats(void)
{
 int i;

 if (trace) fprintf(fpt,"RESET_MR_STATs\n");

 mr_p_c=0;
 mr_p_c_sq=0;
 mr_t_i_s=0;
 mr_t_i_s_sq=0;
 mr_t_o_c=0;
 mr_t_o_c_sq=0;

}

////////////////////////////////////////////////////////////// UPDATE MR STATS

void update_mr_stats(void)
{
 mr_p_c+=parts_com;
 mr_p_c_sq+=parts_com*parts_com;
 mr_t_i_s+=avg_time_syst;
 mr_t_i_s_sq+=avg_time_syst*avg_time_syst;
 mr_t_o_c+=avg_t_o_c;
 mr_t_o_c_sq+=avg_t_o_c*avg_t_o_c;

}

////////////////////////////////////////////////////////////// RESET REP UTILS

void reset_utils(void)
{
 int i;

  if (trace) fprintf(fpt,"RESET_UTILIZATIONS\n");

  for (i=0;i<11;i++)
   {
     util_sta[i] =0.0;
    // util_sta_sq[i] =0;
   }
```

```
  util_conv=0.0;
}

////////////////////////////////////////////////////////////// RESET MR UTILS

void reset_mr_utils(void)
{
 int i;

  if (trace) fprintf(fpt,"RESET_MR_UTILS\n");

  for (i=0;i<11;i++)
    {
      mr_util_sta[i] =0.0;
      //mr_util_sta_sq[i] =0;
    }
  mr_util_conv=0.0;
}

////////////////////////////////////////////////////////////// UPDATE_UTILIZATIONS

void update_utils(long int curtime, long int oldtime)
{
 long int time;
 int i;
 event* temp;

 if (trace) fprintf(fpt,"UPDATE_UTILIZATIONS\n");

 time=curtime-oldtime;

 temp=calender1;

 while (strcmp(temp->name,"end") != 0)
   { if (!temp->isempty)
        util_conv+=time;

     temp=temp->next;
   }

   for (i=0; i<=num_machines+separate_unloader; i++)
       {
         if (station[i].busy)

           util_sta[i]+=time;

       }

}

//////////////////////////////////////////////////////////////UPDATE MR UTILS

void update_mr_utils(void)
{
 int i;

 mr_util_conv+=util_conv;
 for (i=0; i<num_machines+separate_unloader+1; i++)
   {
     mr_util_sta[i]+=util_sta[i];
```

```
    }
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                          OUTPUT MODULES                            //
//                                                                    //
////////////////////////////////////////////////////////////////////////




int digitcount(int numb)
{
int ret;
ret=1+(numb>9)+(numb>99)+(numb>999)+(numb>9999);
return ret;
}

////////////////////////////////////////////////// PRINT CURRENT CONFIG FILE

void print_current_config_file(void)
{
  int i,j,c,done,size,ind;
 int done_user,index,perc;

 if (trace) fprintf(fpt,"PRINT_CURRENT_CONFIG_FILE\n");

 fprintf(fp,"System Designer:    %s  %s\n", first_name,last_name);
 fprintf(fp,"Date: %s \n\n",date);
 fprintf(fp,"System Configuration:\n\n");
 fprintf(fp,"Station  Station  Conveyor  Robot time                ");
 fprintf(fp,"Service time\n");
 fprintf(fp,"number   name     /Pallet   distribution              ");
 fprintf(fp,"distribution\n");
 fprintf(fp,"                  behavior  (in seconds)              ");
 fprintf(fp,"(in seconds)\n");
 fprintf(fp,"_______  _______ ________  ____________              ");
 fprintf(fp,"____________\n\n");

 for (i=0; i<=num_machines+separate_unloader; i++)
   {
     if (i<10)
       fprintf(fp," %d       %s",i,station[i].name);
     else
       fprintf(fp,"%d       %s",i,station[i].name);

     size=strlen(station[i].name);
     for (c=1; c<=9-size; c++)
       fprintf(fp," ");

     if (station[i].conveyorwait)
           fprintf(fp,"   W/W    ");
```

```
else if (station[i].palletwait)
      fprintf(fp,"   C/W    ");

else
      fprintf(fp,"   C/C    ");

if (strcmp(station[i].robot_time,"tria")==0)
  {
      fprintf(fp,"%s(%d,%d,%d)",  station[i].robot_time,
                                  station[i].rtp1,
                                  station[i].rtp2,
                                  station[i].rtp3 );

      size=2+digitcount(station[i].rtp1)+digitcount(station[i].rtp2)+
            digitcount(station[i].rtp3);
  }
else if (strcmp(station[i].robot_time,"norm")==0 ||
         strcmp(station[i].robot_time,"unif")==0 )
  {
      fprintf(fp,"%s(%d,%d)",     station[i].robot_time,
                                  station[i].rtp1,
                                  station[i].rtp2);

      size=1+digitcount(station[i].rtp1)+digitcount(station[i].rtp2);
  }
else if ( (strcmp(station[i].robot_time,"expo")==0) ||
          (strcmp(station[i].robot_time,"cons")==0) )
  {
      fprintf(fp,"%s(%d)",        station[i].robot_time,
                                  station[i].rtp1);

      size=digitcount(station[i].rtp1);
  }
else
  {    fprintf(fp,"%s(",           station[i].robot_time);

       done_user=false;
       index=0;
       perc=0;
       size=0;
       while ( !done_user )
         { fprintf(fp,"%d:%d",    station[i].rtp4[index],
                                  station[i].rtp5[index]);

           perc+=station[i].rtp5[index];

           size++;
           size+=digitcount(station[i].rtp4[index]);
           size+=digitcount(station[i].rtp5[index]);

           if ( perc < 100 )
             {
               fprintf(fp,",");
               size++;
             }
           else
             { done_user=true;
               fprintf(fp,")");
             }
           index++;
```

```
        }

 }

for (c=1; c<=19-size; c++)
  fprintf(fp," ");

if ( (strcmp(station[i].name,"enter")!=0) &&
     (strcmp(station[i].name,"remove")!=0) &&
     (strcmp(station[i].name,"exit" )!=0) )
  {

    if (strcmp(station[i].service_time,"tria")==0)

      fprintf(fp,"%s(%d,%d,%d)",station[i].service_time,
                                station[i].stp1,
                                station[i].stp2,
                                station[i].stp3 );

    else if (strcmp(station[i].service_time,"norm")==0 ||
         strcmp(station[i].service_time,"unif")==0 )

      fprintf(fp,"%s(%d,%d)",station[i].service_time,
                             station[i].stp1,
                             station[i].stp2);

    else if ( (strcmp(station[i].service_time,"expo")==0) ||
              (strcmp(station[i].service_time,"cons")==0) )

      fprintf(fp,"%s(%d)",station[i].service_time,
                          station[i].stp1);
    else
      {
        fprintf(fp,"%s(",           station[i].service_time);

        done_user=false;
        index=0;
        perc=0;
        size=0;
        while ( !done_user )
          {
            fprintf(fp,"%d:%d",    station[i].stp4[index],
                                   station[i].stp5[index]);

            perc+=station[i].stp5[index];

            size++;
            size+=digitcount(station[i].stp4[index]);
            size+=digitcount(station[i].stp5[index]);

            if ( perc < 100 )
              {
               fprintf(fp,",");
               size++;
              }
            else
              { done_user=true;
                fprintf(fp,")");
              }
            index++;
```

```
                }
            }
        }
      fprintf(fp,"\n");

    }
  if (!separate_unloader)
    {
      ind=num_machines+1;
      if (num_machines==9)
        fprintf(fp,"%d       %s",ind,station[ind].name);
      else
        fprintf(fp," %d       %s",ind,station[ind].name);
      fprintf(fp,"    (the load station removes the finished parts)\n");
    }


  fprintf(fp,"_____________________________________________________________");
  fprintf(fp,"__________\n\n");
  fprintf(fp,"Conveyor Length:\t%-d feet.\n",length_of_conveyor);
  fprintf(fp,"Conveyor Speed: \t%-d seconds to complete 1 revolution.\n",
                                                    speed_of_conveyor);
  fprintf(fp,"Pallet load/unload time (C/W only):\t%-ld seconds.\n\n",
                                                    pallet_unload_time);


}

//////////////////////////////////////////// PRINT CURRENT SIMULATION FILE

void print_current_simulation_file(void)
{
  if (trace) fprintf(fp,"PRINT_CURRENT_SIMULATION_FILE\n");

  fprintf(fp,"Maximum Number of Pallets (max parts in system):\t%d\n",
                                                  max_pallets_in_system);
  if (inorder)
     fprintf(fp,"Processing by machines is to be done:     IN THIS ORDER\n\n");
  else
     fprintf(fp,"Processing by machines is to be done:     IN ANY ORDER\n\n");

  print_current_operations_file();

  fprintf(fp,"Number of Replications:\t\t%d\n",reps);
  fprintf(fp,"Length of each replication:\t%ld seconds\t(%ld hours)\n",
                                                    length_of_sim,
                                                    length_of_sim/3600 );
  fprintf(fp,"Warm-up period:\t\t\t%ld seconds\t(%ld hours)\n",
                                                    warm_up,
                                                    warm_up/3600);
  fprintf(fp,"_____________________________________________________________");
  fprintf(fp,"__________\n\n");
}

//////////////////////////////////////////// PRINT CURRENT OPERATIONS FILE

void print_current_operations_file(void)
{
  int j,done;
```

```
 fprintf(fp,"Machining operations required:");
 fprintf(fp,"\t\t      %s\n",mach_seq_reqd[0]);

 done=false;
 j=1;
 while (!done)
   { if (strcmp(mach_seq_reqd[j],"remove")==0)
       {
         fprintf(fp,"\t\t\t\t\t      %s\n\n",mach_seq_reqd[j]);
         done=true;
       }
     else
       {
         fprintf(fp,"\t\t\t\t\t      %s\n",mach_seq_reqd[j]);
         j++;
       }
   }

}

////////////////////////////////////////////////////////// PRINT REP REPORT

void print_rep_report(int rep)
{
 int      i;

 double   s_d,
          s_d_c,
          denom;

 clrscr();

 if(outfile)
 fprintf(fp,"System Designer:    %s  %s\t\t\t", first_name,last_name);

 if(outfile)
 fprintf(fp,"Date: %s \n\n",date);

 if (rep==0) printf("Results of Warm-up Period:\n\n");
 else printf("Results of replication #%d:\n\n",rep);

 if(outfile)
  { if (rep==0) fprintf(fp,"Results of Warm-up Period:\n\n");
    else fprintf(fp,"Results of replication #%d:\n\n",rep);
  }

 printf("    Length of simulation:\t\t\t%ld\n",length_of_sim);

 if(outfile)
 fprintf(fp,"    Length of simulation:\t\t\t%ld\n",length_of_sim);


 printf("    Number of parts completed:\t\t\t%d\n\n",parts_com);

 if(outfile)
 fprintf(fp,"    Number of parts completed:\t\t\t%d\n\n",parts_com);

 printf("    \t\t\t\t\tmin\tmax\tavg\tS.D.\n\n");

 if(outfile)
```

```
fprintf(fp,"    \t\t\t\t\tmin\tmax\tavg\tS.D.\n\n");

denom=parts_com-1.0;
s_d=(total_time_sq-avg_time_syst*avg_time_syst*parts_com)/denom;

printf (   "    Time parts spent in system:\t\t%ld\t%ld\t%6.2f\t%6.2f\n",
           min_t_i_s, max_t_i_s, avg_time_syst, sqrt(s_d) );

if(outfile)
fprintf(fp,"    Time parts spent in system:\t\t%ld\t%ld\t%6.2f\t%6.2f\n",
         min_t_i_s, max_t_i_s, avg_time_syst, sqrt(s_d) );

s_d_c=(total_t_o_c_sq-avg_t_o_c*avg_t_o_c*parts_com)/denom;

printf (   "    Time parts spent on conveyor:\t%ld\t%ld\t%6.2f\t%6.2f\n\n",
           min_t_o_c, max_t_o_c, avg_t_o_c, sqrt(s_d_c) );

if(outfile)
fprintf(fp,"    Time parts spent on conveyor:\t%ld\t%ld\t%6.2f\t%6.2f\n\n",
           min_t_o_c, max_t_o_c, avg_t_o_c, sqrt(s_d_c) );

util_conv=util_conv/fl_length_of_sim;

printf (   "    Average number of parts on conveyor:\t\t%-6.2f\n",util_conv);
printf (   "    Utilizations of stations:\n");

if(outfile)
 {
fprintf(fp,"    Average number of parts on conveyor:\t\t%-6.2f\n",util_conv);
fprintf(fp,"    Utilizations of stations:\n");
 }


for (i=0; i<num_machines+separate_unloader+1; i++)
  { util_sta[i]=util_sta[i]/fl_length_of_sim;

    printf (   "\t\t\t\t\t\t%s:\t\t%6.2f\n",station[i].name,
                                        util_sta[i]);

    if(outfile)
    fprintf(fp,"\t\t\t\t\t\t%s:\t\t%6.2f\n",station[i].name
                                        ,util_sta[i]);
  }

printf (   "\n_____________________________________________________________");
printf (   "_____________________\n");

if(outfile)
fprintf(fp,"_____________________________________________________________");

if(outfile)
fprintf(fp,"_____________________\n\n");

if ((rep==0) ||
    (num_machines>=4 && rep/2==(rep+1)/2) ||
    (num_machines<=3 &&(rep==3 || rep==6 || rep==9 || rep==12 || rep==15
                                          || rep==18 ))||
    (rep==reps) )

  if(outfile)
```

```
   fprintf(fp,"\f");
}

////////////////////////////////////////////////////////PRINT_OVERALLREPORT

void print_overallreport(void)
{
  int i;
  double s_sq,
         f_reps;

 if(outfile)
 fprintf(fp,"System Designer:    %s  %s\n", first_name,last_name);

 if(outfile)
 fprintf(fp,"Date: %s \n\n",date);

 f_reps=(float)reps;
 clrscr();

 printf (   "Overall Results:\n\n");

 if(outfile)
 fprintf(fp,"Overall Results:\n\n");

 printf (   "    Number of replications: \t\t\t\t%d\n",reps);
 if(outfile)
 fprintf(fp,"    Number of replications: \t\t\t\t%d\n",reps);

 printf(    "    Length of each replication:\t\t\t\t%ld\n\n",length_of_sim);

 if(outfile)
 fprintf(fp,"    Length of each replication:\t\t\t\t%ld\n\n",length_of_sim);

 printf(    "    Average Number of parts completed:\t\t\t%-1.2f\n",mr_p_c/f_reps

 if(outfile)
 fprintf(fp,"    Average Number of parts completed:\t\t\t%-1.2f\n",mr_p_c/f_reps


 s_sq=((mr_p_c_sq-mr_p_c*mr_p_c/f_reps)/(f_reps-1.0));

 printf (   "\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq) );

 if(outfile)
 fprintf(fp,"\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq) );

 printf (   "\t95%% C.I. for mean parts completed:\t\t(%1.2f,%1.2f)\n\n",
           mr_p_c/f_reps-(1.2*sqrt(s_sq/f_reps)),
           mr_p_c/f_reps+(1.2*sqrt(s_sq/f_reps)) );


 if(outfile)
 fprintf(fp,"\t95%% C.I. for mean parts completed:\t\t(%1.2f,%1.2f)\n\n",
           mr_p_c/f_reps-(           1.2         *sqrt(s_sq/f_reps)),
           (mr_p_c/f_reps+(           1.2         *sqrt(s_sq/f_reps))));


 printf (   "    Average Time parts spent in system:\t\t\t%1.2f\n",
```

```
                    mr_t_i_s/f_reps);

if(outfile)
fprintf(fp,"    Average Time parts spent in system:\t\t\t%1.2f\n",
                    mr_t_i_s/f_reps);

s_sq=(mr_t_i_s_sq-mr_t_i_s*mr_t_i_s/f_reps)/(f_reps-1.0);


if(outfile)
fprintf(fp,"\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq));

if(outfile)
fprintf(fp,"\t95%% C.I. for mean time in system:\t\t(%1.2f,%1.2f)\n\n",

      mr_t_i_s/f_reps-(          1.2        *sqrt(s_sq/f_reps)),
      mr_t_i_s/f_reps+(          1.2        *sqrt(s_sq/f_reps)));

printf (   "\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq));
printf (   "\t95%% C.I. for mean time in system:\t\t(%6.2f,%6.2f)\n\n",

      mr_t_i_s/f_reps-(          1.2        *sqrt(s_sq/f_reps)),
      mr_t_i_s/f_reps+(          1.2        *sqrt(s_sq/f_reps)));


if(outfile)
fprintf(fp,"    Average Time parts spent on conveyor:\t\t%1.2f\n",
      mr_t_o_c/f_reps);

printf (   "    Average Time parts spent on conveyor:\t\t%1.2f\n",
      mr_t_o_c/f_reps);

s_sq=(mr_t_o_c_sq-mr_t_o_c*mr_t_o_c/f_reps)/(f_reps-1);


if(outfile)
fprintf(fp,"\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq));

if(outfile)
fprintf(fp,"\t95%% C.I. for mean time on conveyor:\t\t(%1.2f,%1.2f)\n\n",

          mr_t_o_c/f_reps-(          1.2        *sqrt(s_sq/f_reps)),
          mr_t_o_c/f_reps+(          1.2        *sqrt(s_sq/f_reps)));


printf (   "\tStandard Deviation:\t\t\t\t%1.2f\n",sqrt(s_sq));
printf (   "\t95%% C.I. for mean time on conveyor:\t\t(%1.2f,%1.2f)\n\n",

          mr_t_o_c/f_reps-(          1.2        *sqrt(s_sq/f_reps)),
          mr_t_o_c/f_reps+(          1.2        *sqrt(s_sq/f_reps)));

mr_util_conv=mr_util_conv/f_reps;

if(outfile)
 {
fprintf(fp,"    Average Number of parts on Conveyor:\t\t\t%-1.2f\n\n",
                                                    mr_util_conv);
fprintf(fp,"    Average Utilization of stations:\n");
 }
printf (   "    Average Number of parts on Conveyor:\t\t%-1.2f\n",
```

```
                                                                mr_util_conv);
 printf (    "    Average Utilization of stations:\n");

 for (i=0; i<num_machines+separate_unloader+1; i++)
     { mr_util_sta[i]=mr_util_sta[i]/f_reps;

       if(outfile)
       fprintf(fp,"\t\t\t\t\t%s:\t\t\t%-1.2f\n",station[i].name,mr_util_sta[i]);

       printf (    "\t\t\t\t\t%s:\t\t\t%-1.2f\n",station[i].name,mr_util_sta[i]);

     }

 if(outfile)
 fprintf(fp,"\f");


}

////////////////////////////////////////////////////////////////// PRINTCALS

void printcals(void)
{
  event* t;

  t=calender1;
  if (trace)
      fprintf(fpt,"PRINTCALS\n");

  fprintf(fpdb,"\tcalender1:\n");
  while (strcmp(t->name,"end") != 0)
    { if (debug)
        fprintf(fpdb,"\t\tPart_num: %d\tName: %s\tSta_num: %d\tTime: %ld\n",
                     t->partnum,t->name,t->stanum,t->time);
      t=t->next;
    }
  t=calender2;
  if (debug)
      fprintf(fpdb,"\tcalender2:\n");
  while (strcmp(t->name,"end") != 0)
    { if (debug)
        fprintf(fpdb,"\t\tPart_num: %d\tName: %s\tSta_num: %d\tTime: %ld\n",
                     t->partnum,t->name,t->stanum,t->time);
      t=t->next;
    }
  fprintf(fpdb,"\n");
}


////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                              TESTCASES                                 //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////// TESTCASE 0

void testcase0(void)
{
  int i,j;

  if (trace) fprintf(fpt,"TESTCASE 0 \n");

  inorder=true;
  length_of_conveyor=40;
  speed_of_conveyor=60;
  separate_unloader = false;
  num_machines=1;
  max_pallets_in_system=2;
  interstation_time=speed_of_conveyor/(num_machines+1+separate_unloader);
  pallet_unload_time=10;

  strcpy(mach_seq_reqd[0],"enter");
  strcpy(mach_seq_reqd[1],"lathe");
  for (i=2; i<11; i++)
     strcpy(mach_seq_reqd[i],"remove");


  strcpy(station[0].name,"enter");

  strcpy(station[0].robot_time,"unif");
  station[0].rtp1=10;
  station[0].rtp2=20;

  strcpy(station[1].name,"lathe");

  strcpy(station[1].robot_time,"unif");
  station[1].rtp1=10;
  station[1].rtp2=20;

  strcpy(station[1].service_time,"norm");
      station[1].stp1=50;
      station[1].stp2=5;

  for (i=0; i<2; i++)
    { station[i].busy=false;
      station[i].done=false;
      station[i].conveyorwait=true;
      station[i].palletwait=false;
    }

}

//////////////////////////////////////////////////////////////// TESTCASE 1

void testcase1(void)
{
 int i,j;

 if (trace) fprintf(fpt,"TESTCASE 1 \n");

 inorder=true;
 length_of_conveyor=40;
 speed_of_conveyor=3;
 separate_unloader = false;
```

```
num_machines=2;
max_pallets_in_system=3;
interstation_time=speed_of_conveyor/(num_machines+1+separate_unloader);
pallet_unload_time=10;

strcpy(mach_seq_reqd[0],"enter");
strcpy(mach_seq_reqd[1],"lathe");
strcpy(mach_seq_reqd[2],"drill");


for (i=3; i<11; i++)
   strcpy(mach_seq_reqd[i],"remove");


strcpy(station[0].name,"enter");

strcpy(station[0].robot_time,"unif");
station[0].rtp1=10;
station[0].rtp2=20;

strcpy(station[1].name,"lathe");

strcpy(station[1].robot_time,"unif");
station[1].rtp1=10;
station[1].rtp2=20;

strcpy(station[1].service_time,"norm");
    station[1].stp1=50;
    station[1].stp2=5;

strcpy(station[2].name,"drill");

strcpy(station[2].robot_time,"unif");
station[2].rtp1=10;
station[2].rtp2=20;

strcpy(station[2].service_time,"norm");
    station[2].stp1=100;
    station[2].stp2=10;

for (i=0; i<3; i++)
  { station[i].busy=false;
    station[i].done=false;
    station[i].conveyorwait=true;
    station[i].palletwait=false;
  }
}

////////////////////////////////////////////////////////////// TESTCASE_2

void testcase2(void)
{
 int i,j;

 if (trace) fprintf(fpt,"TESTCASE 2 \n");

 inorder=true;
 length_of_conveyor=60;
 speed_of_conveyor=5;
 separate_unloader = false;
```

```
num_machines=3;
max_pallets_in_system=4;
interstation_time=speed_of_conveyor/(num_machines+1+separate_unloader);
pallet_unload_time=10;

strcpy(mach_seq_reqd[0],"enter");
strcpy(mach_seq_reqd[1],"lathe");
strcpy(mach_seq_reqd[2],"drill");

for (i=3; i<11; i++)
   strcpy(mach_seq_reqd[i],"remove");


strcpy(station[0].name,"enter");

strcpy(station[0].robot_time,"norm");
station[0].rtp1=3;
station[0].rtp2=5;

strcpy(station[1].name,"lathe");

strcpy(station[1].robot_time,"norm");
station[1].rtp1=30;
station[1].rtp2=5;

strcpy(station[1].service_time,"unif");
    station[1].stp1=300;
    station[1].stp2=60;

strcpy(station[2].name,"drill");

strcpy(station[2].robot_time,"norm");
station[2].rtp1=30;
station[2].rtp2=5;

strcpy(station[2].service_time,"tria");
    station[2].stp1=10;
    station[2].stp2=14;
    station[2].stp3=16;

strcpy(station[3].name,"lathe");

strcpy(station[3].robot_time,"norm");
station[3].rtp1=80;
station[3].rtp2=5;

strcpy(station[3].service_time,"unif");
    station[3].stp1=300;
    station[3].stp2=60;

for (i=0; i<4; i++)
  { station[i].busy=false;
    station[i].done=false;
    station[i].conveyorwait=false;
    station[i].palletwait=false;
  }
}

////////////////////////////////////////////////////////////// TESTCASE_3
```

```
void testcase3(void)
{
 int i,j;

 if (trace) fprintf(fpt,"TESTCASE 3 \n");

 inorder=true;
 length_of_conveyor=60;
 speed_of_conveyor=5;
 separate_unloader = false;
 num_machines=3;
 max_pallets_in_system=4;
 interstation_time=speed_of_conveyor/(num_machines+1+separate_unloader);
 pallet_unload_time=10;

 strcpy(mach_seq_reqd[0],"enter");
 strcpy(mach_seq_reqd[1],"lathe");
 strcpy(mach_seq_reqd[2],"drill");

 for (i=3; i<11; i++)
    strcpy(mach_seq_reqd[i],"remove");


 strcpy(station[0].name,"enter");

 strcpy(station[0].robot_time,"norm");
 station[0].rtp1=3;
 station[0].rtp2=5;

 strcpy(station[1].name,"lathe");

 strcpy(station[1].robot_time,"norm");
 station[1].rtp1=30;
 station[1].rtp2=5;

 strcpy(station[1].service_time,"unif");
     station[1].stp1=300;
     station[1].stp2=60;

 strcpy(station[2].name,"drill");

 strcpy(station[2].robot_time,"norm");
 station[2].rtp1=30;
 station[2].rtp2=5;

 strcpy(station[2].service_time,"tria");
     station[2].stp1=10;
     station[2].stp2=14;
     station[2].stp3=16;

 strcpy(station[3].name,"lathe");

 strcpy(station[3].robot_time,"norm");
 station[3].rtp1=80;
 station[3].rtp2=5;

 strcpy(station[3].service_time,"unif");
     station[3].stp1=300;
     station[3].stp2=60;
```

```
 for (i=0; i<4; i++)
   { station[i].busy=false;
     station[i].done=false;
     station[i].conveyorwait=false;
     station[i].palletwait=true;
   }

}
```