Ferry Watch Scheduling Prototype and Recommended Future Work

WA-RD 241.1

Final Report
March 1992

Washington State Department of Transportation
Washington State Transportation Commission
in cooperation with the
United States Department of Transportation
Federal Highway Administration

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO. WA-RD 241.1	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
4. TITLE AND SUBTITLE FERRY WATCH SCHEDULING PROTOTYPE AND RECOMMENDED FUTURE WORK		5. REPORT DATE March 1992
		6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) Mark E. Hallenbeck and Jua-Been Chang		8. PERFORMING ORGANIZATION REPORT NO.
9. PERFORMING ORGANIZATION NAME AND ADDRESS Washington State Transportation Center (TRAC) University of Washington, JE-10 The Corbet Building, Suite 204; 4507 University Way N.E. Seattle, Washington 98105		10. WORK UNIT NO.
		11. CONTRACT OR GRANT NO. GC8719, Task 10
12. SPONSORING AGENCY NAME AND ADDRESS Washington State Department of Transportation Transportation Building, KF-01 Olympia, Washington 98504		13. TYPE OF REPORT AND PERIOD COVERED Final report
		14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

This study was conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.

16. ABSTRACT

This report documents the initial exploration of alternatives for a computer system that assists in the development of watch schedules for the Washington State Ferry System (WSF). A "watch schedule" is defined as a two-week set of work shifts to be followed by a group of WSF employees from a specific union. The report describes the programing alternatives considered, the program flow selected for prototype development, and the conclusions and recommendations drawn from the creation of that prototype. Continued development of the watch scheduling system is not recommended at this time.

17. KEY WORDS Ferry System Operations, Crew Scheduling, Labor Scheduling	18. DISTRIBUTION STATEMENT No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None	78	

Final Report

Research Project GC 8719 Task 10
Ferry Crew (Watch) Scheduling

FERRY WATCH SCHEDULING PROTOTYPE AND RECOMMENDED FUTURE WORK

by

Mark Hallenbeck
Senior Research Engineer

Jua-Been Chang
Research Assistant

Washington State Transportation Center (TRAC)
University of Washington, JE-10
The Corbet Building, Suite 204
4507 University Way N.E.
Seattle, Washington 98105

Washington State Department of Transportation
Technical Monitor
Ray Deardorff
Director of Planning

Prepared for

Washington State Transportation Commission
Department of Transportation
and in cooperation with
U.S. Department of Transportation
Federal Highway Administration

March 1992

DISCLAIMER

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This report documents the initial exploration of alternatives for a computer program to develop watch schedules for the Washington State Ferry System. (A "watch schedule" is defined as a two-week set of work shifts to be followed by a group of WSF employees from a specific union.) The report presents

- a short introduction to watch scheduling,
- a review of the current procedures used by the WSF,
- a description of the alternative system designs considered for this project,
- a detailed description of the program flow for the alternative selected for prototype development, and
- the conclusions and recommendations drawn from the development of that prototype.

Finally, the report's appendices contain directions for converting the program flow described in the fourth chapter into source code, the source code for the prototype, and programing insights gained from the prototype development. A diskette containing the compiled program code has been sent to the WSF planning office.

BACKGROUND

The WSF has had a reasonably stable route system for many years. Vessel schedules operating on these routes are fairly constrained by existing labor rules and legislative restrictions on the levels of service that WSF can provide. These limitations have resulted in vessel schedules that have not changed dramatically over the last few years.

Because the vessel schedules have been consistent, the WSF has been able to use historical watch schedules as the basis for new watch schedules. These new schedules are reasonably efficient, although they contain some unproductive time (time when crews assigned to watches are paid but are not actively involved in operating a vessel in revenue service). Unproductive time is primarily caused by WSF's inability to match vessel and personnel schedules within the constraints of the existing labor agreements and passenger requirements.

WSF currently develops crew schedules by hand. In most instances, WSF personnel simply use the watch schedules that were developed for previous vessel schedules when those schedules were in operation. When vessel schedules change significantly (for example, with the addition of the passenger only vessels), WSF staff must manually develop the appropriate watch schedules.

This method for scheduling personnel works acceptably because of the similarity of current vessel schedules and labor rules to historical schedules and rules. However, if either the union rules or a large portion of the vessel schedules were to change dramatically, this manual process would be difficult to use and prone to error and inefficiency.

WSF has not significantly investigated radical new vessel schedules or changes in labor rules partially because of the large cost and staff time involved in using this manual method to test those changes. The investigation into a computer aided watch scheduler was started partly to examine whether significant cost savings or service improvements could be obtained from these types of changes.

PREVIOUS STUDIES

The WSF started the review of its scheduling systems in 1985. The report, "Ferry Systems Data, Scheduling and Billing, Scheduling Systems Analysis," was completed in June 1987. This report examined the need for, and advantages to be gained by, computerizing the vessel and crew scheduling procedures and looked at the possibility of obtaining this software in four different ways: writing custom software, using public domain scheduling software, modifying public domain software to meet WSF needs, or purchasing commercially developed scheduling software designed for either the railroad or airline industries.

The 1987 study found that roughly seven percent of the labor costs ($39,400 per two-week period) for the Fall 1986 schedule were spent for labor outside of the operation of the scheduled service. The majority of these costs were unnecessary labor expenses. They were incurred when crews had no assigned duties during some part of their paid work week and when a watch from a larger vessel was used to crew a smaller vessel, thus exceeding the necessary crew size mandated by the labor agreements and Coast Guard regulations.

This study also determined that the WSF could reduce staffing costs by scheduling individuals, rather than keeping each watch together for the entire two-week work schedule. However, this concept (called scheduling by position) would greatly increase the complexity of the scheduling process and might cause labor relations problems because it would deviate significantly from the current procedures.

The report concluded that none of the existing scheduling software would meet the WSF's needs "off-the-shelf." It recommended that the WSF consider either developing its own software system or purchasing one from a commercial vendor, who would modify an existing package to meet the WSF's needs.

As a result of the Scheduling Systems work described above, the WSF did develop a computer aided vessel scheduling system. This system is now used by WSF and has been very well received. Upon the successful implementation of the vessel scheduler, WSF authorized the development of a prototype watch scheduler to gain a better understanding of the complexities of the watch scheduling problem and to better determine whether the cost of such a system was justified.

POTENTIAL USES FOR COMPUTER-ASSISTED WATCH SCHEDULING

To start this effort, the project team examined the various tasks that a computer aided watch scheduler might accomplish and tried to determine which tasks provided the greatest opportunity for the WSF. The following tasks were identified as areas in which a computer aided scheduler could provide significant improvements over the existing manual process:

- the investigation of alternative work rules for future labor negotiations,
- the investigation of the impact of radically different vessel schedules on labor usage,
- the routine creation of watch schedules, and
- the implementation of scheduling by position.

Each of these tasks would require a different mix of capabilities for the scheduling system. Each of these tasks is discussed below.

Testing Labor Contract Changes

A tool that could help determine the feasibility and costs of different staffing rules could lead to a more dynamic bargaining process in the labor negotiations WSF undertakes with its various unions. Changes in labor rules have the potential to provide significant increases in ferry service at relatively modest cost. To obtain these savings, WSF may need to provide increased wages or other concessions to the labor unions. Before granting concessions, WSF must fully explore the implications of proposed rule and wage changes in terms of the service enhancements provided, the size of the labor force required to provide that service, and the total cost of that service.

The only way to adequately make these comparisons is to develop vessel and watch schedules that use the proposed work rules. The existing computer aided vessel scheduler makes the creation of the vessel schedules easy, but the WSF does not have the ability to easily create watch schedules because of the complexity of the labor rules. This inability slows the analysis process and often causes labor rule change to be discarded from the negotiation process because the implications of the changes can not be determined within the time frame of the negotiations.

The service changes that might be possible, given changes in the existing labor contracts, are as follows:

- an increase in peak service, oriented towards commuters, through the operation of some vessels only during the peak hours,
- longer service hours for the heavily used recreational routes (i.e., increasing an operation from 16 hours daily to 20 hours), and
- the creation of new routes and services.

To be economically feasible, these service changes would require significant revision in the labor rules.

To be used for the above analyses, the scheduling system would have to be flexible enough to incorporate a wide variety of labor rules that are not currently legal at the WSF. Labor rules at various other transit authorities that might benefit WSF include the following:

- split shifts, to accommodate commuter traffic,
- different shift length and day off combinations, for example, a ten-hour work day with a four-day work week,
- the ability to schedule individuals rather than entire watches,
- increased use of part-time workers,
- scheduled overtime, and
- special wage increases for overtime, spread time, and shift differentials.

This group of labor rules has a variety of alternative programing implications. A programing technique that is appropriate for testing alternative shift lengths is not necessarily useful for looking at split shifts and the introduction of spread time. In addition, applying all of these rules means that the computer operator must provide the computer with the necessary input information for each alternative and inform the computer program about the alternatives to include in any one program run.

As a result, these potential rule changes would make the scheduling process and the computer program that replicated that process very complex. This complexity would be present in the design and programing of the system, and in the knowledge the computer operator would have to operate the computer program. Thus, to develop a scheduling assistant for these tasks would be expensive, and the program used would require fairly intensive training.

Testing Service Changes

Computer assistance in the watch scheduling process could potentially improve the Ferry System's ability to rapidly create new crew schedules in response to new routes and major service changes. While these types of service changes are not expected soon, the continued growth in ferry ridership raises the probability that additional vessels and possibly new routes will be added to meet the growing demand. These additions might create situations in which the historical watch schedules were no longer valid, and watch schedules would have to be developed from scratch.

The computer aided scheduler might be useful for quickly creating new crew schedules. This would allow more responsive planning and would also allow the WSF to examine more alternatives than would otherwise be possible.

To obtain these advantages, the computer system would have to be able to apply existing labor rules, and it would have to provide a significant speed advantage over the existing manual system. While the features listed above for testing new labor rules would be useful, they would be unnecessary if they complicated the model so much that they slowed the system down or made it too difficult to easily use.

A computer system that met these limited needs would be easier to build than the program described previously. The program would also operate much more quickly because it would not need to compute and analyze labor use combinations associated with potential labor rule changes.

However, if the computer system was not designed to accommodate the labor rule changes as proposed in the previous section, and one or more of those changes was later adopted by the WSF, a major rewrite of the computer software would be needed to retain the usability of the computer program. This rewrite would be costly because it would require a new programer to learn the existing coding process well enough to revise it at some later date.

If labor changes are likely in the next five years, it would bc far better to build the computer system to handle potential labor rule changes than to limit the initial programing effort just to reduce the cost of the system development, only to have the system become obsolete in several years and require an expensive upgrade.

Routine Scheduling

Computer assistance in the watch scheduling process could potentially improve the Ferry System's ability to rapidly create new crew schedules in response to routine vessel schedule changes and to test the implementation of new routes and service changes. The two computer program alternatives mentioned above would both meet the needs for routine scheduling, but both systems might be more complex than necessary to provide benefits to WSF.

The project team determined that, under this scenario, a computer aid less complex than a full scheduling system might provide benefit to the WSF. One of the most difficult challenges in watch scheduling is simply keeping track of the movement of crews between routes. If a vessel schedule is changed on one route, watches taken from that route to crew vessels on other routes may no longer be within union regulations. Similarly, watches transferred onto the new route may no longer be within legal work limits. In either case, the scheduler must carefully review the work assignments of all watches affected by the change, as well as all watches that might be used to most cost effectively fill (or need filling by) watches on the affected route.

A computer system would be useful for assisting in this function by tracking which watches were available to move between routes and when they could legally be used on other routes. It could also track which watches had already been assigned to other routes and the cost associated with those transfers. When revising a schedule, the planner could then quickly determine whether the changes being considered were feasible from a labor usage standpoint, and the implications of those changes. The planner would also be responsible for updating the computer system for each scheduling change.

Finally, the computer system could also keep track of the cost of transporting watches between routes to assist the planner in selecting between alternative crewing options.

Scheduling By Position

This final function for a scheduling aid would help the WSF decrease the number of occasions when more crew members are being used than the vessel needs. This problem commonly occurs when the crew from a larger vessel is used to operate a smaller vessel. Because each section of a watch (deck crew, engine room crew, and the Masters and Mates) is scheduled as a group, all members of a watch are assigned to the smaller vessel, even though they are not necessary. This costs the WSF some unnecessary crewing wages.

While the 1987 study concluded that it was not in the best interests of the WSF to schedule by position, computer aided scheduling software could provide that capability, should WSF change its mind. The computer system discussed at the beginning of this section could handle scheduling by position with only minor changes. Those changes would center on increasing the number "positions" required for each vessel from one watch to the required number of able seamen, ordinary seamen, and other positions. This means a little more complexity in the user interface portion of the program and a greater computer memory requirement, but neither of these needs would be significant.

ADDITIONAL SYSTEM REQUIREMENTS

In addition to the basic scheduling functions mentioned in the preceding section, a number of other requirements for the scheduling system were mentioned in the 1987 preliminary study. That study identified the following capabilities a watch scheduling system should provide, regardless of the basic functioning of the system.

- The program must be able to generate line-by-line crew schedules.
- The program must be able to generate day-by-day shift assignments (e.g., relief times, place of relief, hours worked, position number(s) worked, days off, and open days).
- The program must be able to generate basic schedule statistics, (i.e., the total number of crew members or shifts required by worker classification, total pay hours over a two-week period, amount of travel pay incurred, number of part time, straight and split shifts).

CHAPTER 2
WSF WATCH (CREW) SCHEDULING

This chapter provides a brief review of the watch scheduling procedures WSF currently follows. It also describes the primary labor rules that impact the development of those schedules and that must be incorporated into any scheduling system.

CURRENT WSF PRACTICE

As indicated in the first chapter, all watch scheduling is currently performed by hand and follows a system that has persisted for many years. On each vessel, three groups of workers (Masters and Mates, the deck crew, and below deck engineers) are assigned shifts over a 14-day pay period. Each of these groups is assigned as a unit, rather than by specific crew position. In other words, the Master and Mate are scheduled as a unit, rather than two separate positions. Each unit consists of all of the personnel required to perform those functions on the vessel. A group of individuals with the same work schedule is referred to as a "watch."

By contract agreement, each full-time crew member must be paid for 80 hours of work during each two-week period. Thus, the WSF must schedule each of these persons for as close to 80 hours per two-week period as possible. In the simplest kind of schedule, each crew member works five, eight-hour shifts each week and has two days off. Unfortunately, this type of schedule severely limits the hours when vessels can be cost-effectively operated. Thus, additional rules have been adopted that both allow WSF management the flexibility to schedule crews for work periods other than five, eight-hour days and protect crew members from unreasonable demands on their time.

Below-Deck Crew

Engineers (members of the MEBA union) generally work 12-hour shifts for seven consecutive days and take the following seven days off. This yields an 84-hour two-week work schedule. By negotiated agreement, the engineers are paid for 80 hours and receive the extra four hours as "comp time." Engineers may also work traditional eight-hour shifts for five consecutive days with two days off. They may not work 10 consecutive days (eight hours each day) followed by four days off.

When a vessel operates a 24-hour schedule, the engineers spend the entire shift operating the vessel engines. When a vessel does not operate in revenue service for all 24-hours, the engineer on duty usually performs routine maintenance on the engine. This full time use of the engineering crew means that no engineers have "dead time" in their schedules, provided that the non-revenue service time is used productively.

Engineers are assigned to specific vessels rather than specific trips or routes. They must also begin and end their work shifts at specific terminals.

Above-Deck Crew

The WSF is not allowed to use the 12-hour per day scheduling system for above-deck personnel (the Master, Mates and Pilots Union (MM&P) or the International Boatmen's Union (IBU) crew). However, neither is WSF constrained to using traditional five-day per week, eight-hours per day schedules. Above-deck workers may work as little as seven and as many as nine hours on any given day without earning overtime pay, as long as they work no more than 80 hours over a two-week period. These crew members may work five days on and two off, or they may work 10 days on and take four consecutive days off. They may also work a "touring watch," in which they work an afternoon shift on one day, sleep on the vessel that night, and work the morning shift the next day, followed by roughly 36 hours off. (This schedule is routinely followed on vessels that serve the Sydney, B.C., terminal.)

Currently, WSF crews are not allowed to work 10-hour per day, four-day per week shifts, except on unscheduled extra service. In addition, the current labor rules limit the WSF to the use of ten part-time (less than 40 hours per week) above-deck personnel and prohibit the use of split shifts. Finally, the current labor contract states that WSF can not intentionally schedule overtime. If a vessel operates late on a given day, overtime is paid as double time, but the WSF can not intentionally schedule crews to operate vessels for more than 80 hours per week.

IMPACT OF LABOR RULES ON VESSEL SCHEDULES

The WSF planning staff may use any or all of these labor rules when developing watch schedules needed to crew the adopted vessel schedules. The difficult part of this task is determining which legal shift types should be applied to which vessel runs, so that the least labor costs are expended to operate the scheduled service.

The prohibition on scheduled overtime and the need to average 8-hour days for above-deck crews (MM&P and IBU) have a significant impact on the schedules that the WSF planning section creates. To make cost-effective use of the available above-deck crew, vessel schedules usually operate for eight, 16 or 24 hours each day. Extending a vessel's routine workday to any length greater than a multiple of eight hours requires that labor be paid for eight hours for any fraction of hours over eight. (For example, an 18-hour schedule requires three separate watches each day, two watches working eight-hour shifts, the third working a two-hour shift. However, the third gets paid for eight hours of work, the minimum pay for a day.) The high marginal cost for these last hours severely limits WSF's ability to add small increments of service to meet growing ridership.

As indicated above, in some cases a shift may operate for nine hours on a given day, but the crew (watch) working that shift must then work a seven-hour shift to stay within the 80-hour, two-week limit. Thus, the combination of vessels operated by that watch must operate daily for a multiple of 8-hours.

It is possible for the WSF to manipulate the nine-hour/seven-hour rule to a limited extent to extend the service day of one vessel while shortening the service day of a second vessel, but these service improvements are extremely limited. The physical reality of scheduling vessels limits the number of locations where this strategy can be employed. Even with the nine-hour/seven-hour service day, the above-deck watch schedule often produces cost inefficiencies that result from an inability to exactly match the crewing requirements to the vessel schedules. (That is, the number of shifts that must be crewed is not evenly divisible by the number of full-time crews.) This is illustrated by the following two examples.

Over a two-week period, a 16-hour schedule requires 28 eight-hour shifts, and a 24-hour schedule requires 42 shifts. Because a crew works for ten shifts each 14 days, two additional shifts need to be filled for a 16-hour boat.

28 shifts - (3 watches * 5 shifts/wk/watch * 2 wks) = -2 shifts

and a 24-hour boat leaves one crew that must be shifted elsewhere for two shifts in a two-week period

42 shifts - (4 watches * 5 shifts/wk/watch * 2 wks) = 2 shifts

Where the extra crew on the 24-hour boat can be shifted to fill the two unfilled shifts on a 16-hour boat, the schedule can be filled without wasting staff resources.

Unfortunately, this match does not always occur, and often in those cases where these matches are possible, the 16- and 24-hour vessels operate on different routes. In these cases, the WSF must pay travel costs to the crews to move between terminals. In fact, most of the day watches on the Bremerton route are filled with "extra" crews from other routes.

Determining which watches should move between routes, which labor rules to use, and how the crews from different watches interact would be the major functions of the proposed software system.

CHAPTER 3

ALTERNATIVE PROTOTYPE DESIGNS

A series of alternative forms for the prototype vessel scheduler were examined as part of this project. Each of the alternatives had a different set of strengths and weaknesses, and each was aimed at fulfilling one or more of the WSF requirements outlined in Chapter 1. For the most part, these systems differed in the complexity of the computer program that would have to be developed and the knowledge and interaction required by the WSF user. The alternatives that were identified are as follows:

- a computer assisted manual process, with the computer performing mathematics and record keeping,
- a computer system operating without manual intervention using a "brute force" computational technique,
- a computer system operating without manual intervention using a specialized algorithm to reduce processing requirements,
- a simplified manual/computer system, and
- alternative ideas.

Each of these alternatives is discussed in more detail below.

A COMPUTER ASSISTED MANUAL PROCESS

This alternative is similar to that developed in the June 1987 analysis of WSF scheduling needs. This alternative is intended to meet all of the requirements stated in the previous chapters. The basic design is a computer aid that is capable of performing the entire scheduling process independent of human input, but it is intended to function best in coordination with human intervention. That is, the WSF service planner would provide input that reduced the number of alternative schedules the computer system would have to evaluate and provide the human insight that is hard to duplicate in a computer program.

This computer system design assumes that the computer would track all scheduling information, but that a WSF planner could accept, reject, or modify all decisions made by the software. This design would free the planner from performing the mathematics and record keeping of the scheduling process, while maintaining the planner's insight in the scheduling process.

The computer system would contain all of the factors necessary for developing schedules. This means it would be complex enough to maintain all of the intricate labor union rules, as well as flexible enough to add rules that WSF would consider in the future.

The concern with this alternative is that by trying to meet all of the needs for a watch scheduler, it would meet none of those needs very well. This is not because of expected limitations in the program design but because some of the "needs" of the desired system appear to conflict with each other. Essentially, by being comprehensive, the system would become complex. Because of the flexibility to change each of the scheduling decisions, the planner operating the system would have to consciously make decisions that were previously made routinely. This would likely slow the scheduling process, and could possibly eliminate the speed advantage computerization was intended to provide.

If the system had little or no speed advantage for the routine scheduling needs of the WSF, it is unlikely that the system would be used for those purposes. Unfortunately, the preliminary investigation of this alternative indicated that this type of system might have a steep learning curve. If the program was not used routinely, this learning curve would be a hindrance to using the system for the more complex (but less common) problems for which the computer would provide the greatest advantage. The project team fears that the result would be that the WSF would continue its current practice of not performing the analyses that would be useful in labor negotiations rather than relearn the computer program.

BRUTE FORCE COMPUTER

The second option for a scheduling system would be to create a computer program that did not need human intervention. By reducing the need for human intervention, the requirement for WSF planner time would be significantly improved. This should result in routine use of the computer system. It would also result in a savings to the WSF in labor costs required to perform the vessel scheduling.

Like the first alternative, this alternative is intended to meet all of the requirements stated in the previous chapters. Also like the first option, this computer system would be designed to contain all of the factors necessary for developing schedules. This means it would be complex enough to maintain all of the intricate labor union rules, as well as flexible enough to add rules that WSF considered in the future. The major difference in this system is that the entire watch scheduling process would be independent of human input.

This particular version of the "all computer" option would use a technique in which the computer determined all legal crewing options and then used a numeric function to determine which of these options was "best." This technique would require large quantities of computation time. The long execution time would slow the system down, but because the system would be designed to operate on a microcomputer with little or no human oversight, the "cost" of this computational time would be minor.

However, without some human oversight, the results from the computer program might or might not be reliable. Historically, computer based schedulers based purely on mathematics (without human review and input) have not produced schedules that were as "good" as human/machine schedulers. This is because the scheduling process is so complex that the algorithms used are not capable of replicating the creative thought process of a good human scheduler. Computer algorithms are also subject to false optimal points that "trick" the software into believing an optimal solution has been found.

Under this scenario, the quality of the watch schedule would rely entirely on the capabilities of the algorithm used by the computer program. A reliable computer program can be written to perform these tasks for under $300,000. The prime difficulty with this effort would be in trying to make the computer software sufficiently flexible that it could be used to look at all reasonable future labor rules.

A secondary drawback might be the WSF staff's reaction to the scheduler output. Computerized schedulers working without human input often create "legal" schedules that are not intuitively obvious. They often do not take into account such factors as "tradition" and local habit. While these factors may not result in "better" schedules, their consideration does provide credibility to the scheduling effort, and without their consideration, schedules are often viewed with some reservation by the personnel who must operate those schedules.

SPECIALIZED COMPUTER ALGORITHM

The third alternative is similar to the second, except that the computer program for this alternative would use one or more specially developed algorithms to reduce the number of scheduling options that would have to be computed and compared. The literature search did not yield any algorithms that would be appropriate for the WSF application without considerable modification and adaptation. However, using "rules of thumb" (much like a human would), it would be possible to reduce the number of possible options without degrading the quality of the program output.

Decreasing the number of solutions the program would have to check would decrease the total processing time and thus improve the operation of the program. Unfortunately, the use of these algorithms could increase the risk of obtaining non-optimal schedules. In addition, the risk of producing non-optimal solutions would increase if the "rules of thumb" applied to reduce the calculation time conflicted with new labor rules added at a later date. (For example, a "rule of thumb" might state that, with the exception of the first trip of the day, a shift could not start earlier than six hours after the start of a vessel's daily schedule. This rule would eliminate the need for the computer to check watch schedules which were not cost effective if eight- or ten-hour shifts were being examined. However, if four-hour shifts were made legal, such a rule would prevent the computer from looking at part-time shifts that started in the morning.)

This alternative would have much the same set of advantages and disadvantages as the brute force alternative. It would have the added advantage of faster execution time, but there would be an increased chance of sub-optimal results.

SIMPLIFIED SYSTEM, PART MANUAL - PART COMPUTER

The fourth alternative is to develop a simplified computer program that is not as powerful as the initial three systems but is easier to use. The intended system would perform the mathematics and accounting described in the first option, but without the complex decision making required for interpreting the labor rules. In this option, the computer program would be more like a spreadsheet that calculated the next legal split times, given a shift starting time and a shift length. The planner using the system would then accept or modify that shift end, and the computer would update when that watch was available again for assignment.

Once shift times were determined, the planner would decide how watches would be divided between shifts and routes. The computer would be used to track available crews and compute travel costs, mostly through automated look-up tables. All "optimization" performed by the system would be performed by the human operator. In this scenario, the computer would only perform data tracking and simple math computations.

It is not clear whether the functions performed by such a system would be worth the development costs of a separate project. It appears from preliminary work that this type of system could evolve more effectively from a WSF planner's normal work with the schedules, rather than from a special research project. The advantage of this is that the planner who worked intimately with the schedules would be able to more accurately determine where the emphasis of the system design should be placed, so that the most benefit could be gained from the programing effort.

ALTERNATIVE IDEAS

The limitations in each of the first four alternatives and the relative lack of interest in those solutions from WSF personnel led the project team to the conclusion that a computer aided watch scheduler may not be the answer to the basic scheduling problems facing the WSF. The majority of the WSF's needs, and most of the potential cost savings, would not be from the routine scheduling of tasks, but from the investigation of advantages that could be gained from alternative work rules.

The project team suggests that as a fifth alternative, rather than continuing to develop a computer program, WSF perform a study that directly investigates the potential impacts of alternative crew rules. The consultant for this proposed study could use the existing vessel scheduling software and any other tools available to perform this analysis. Additional tools (such as those envisioned in the fourth alternative above) might also be developed as part of the project.

The goals of the project would be to determine

- the advantages and disadvantages the proposed work rule changes would provide,
- the conditions under which proposed work rules would be advantageous to WSF,

- the impacts those rules might have on the service levels the WSF could provide, and
- the impact those rules would have on the union members.

A secondary activity of the research might be to develop watch schedules with the proposed union rules that WSF could use after the new rules had been adopted.

Given the stability of the WSF vessel schedules, a careful analysis along these lines would likely address the WSF's needs for many years. Consequently, the results of the proposed study would be useful to WSF even if those results are not acted upon in the near future.

CHAPTER 4
PROTOTYPE DESCRIPTION

From the alternatives discussed in the previous chapter, the project team selected the first alternative, A Computer Assisted Manual Process, for development of the prototype under this contract. This alternative was selected for the prototype development because it would yield the most insight into the complexity of the scheduling problem. By developing this prototype, the project team could gain a better feel for

- the level of human interaction that was appropriate for the scheduling system,
- the time and cost required to develop the decision making and optimization algorithms,
- the information required by the computer system, and
- the training required to learn and operate the system.

To develop the prototype, the project team created a flow diagram of the computer system (see Figure 1) and then wrote the input screens and preliminary processing algorithms that allowed them to explore the use of the program as it was designed.

This chapter outlines the prototype watch scheduling package developed under this contract. More specific programing instructions for converting the program flow description given below into a computer program can be found in the appendix.

INITIAL MENUS

The watch scheduling program would be used as a companion to the vessel scheduling program produced for WSF by Washington State Transportation Center (TRAC) in 1988. Because the vessel scheduling package creates vessel schedules for individual routes on a specific day (separate schedules may be prepared for weekends or holidays, for example), and a watch scheduling program would create schedules for a set

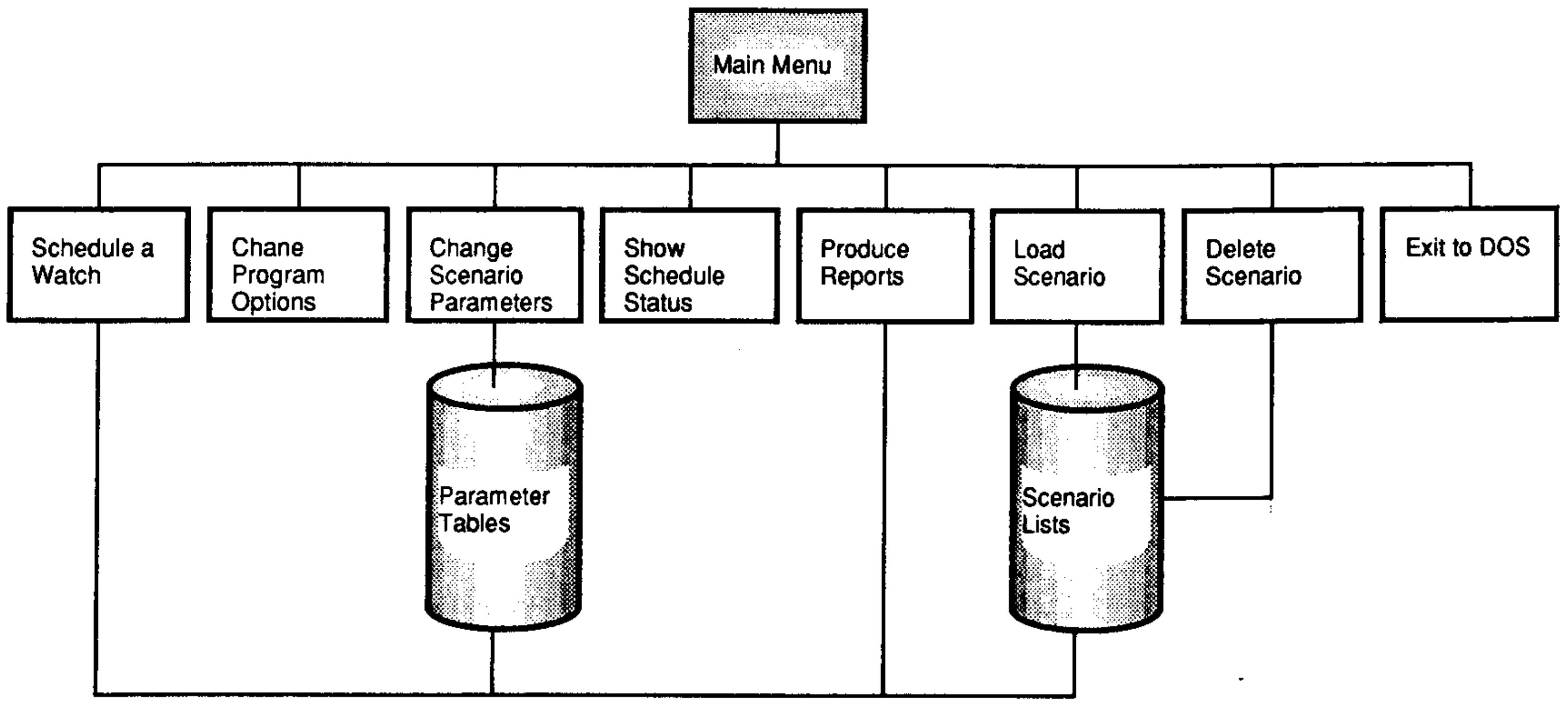


Figure 1. Flow of Prototype Ferry Watch Scheduler

of routes over a two-week period, the watch scheduling program would first need to aggregate a set of vessel schedules so that they became a watch scheduling "scenario." Upon entry into the program, the user would be presented with a menu of currently defined scenarios, as well as the option to create a new scenario (see top of Figure 2). The user might wish to simply load an existing scenario and alter it, or to create a new systemwide or partial system watch schedule from scratch. To define a scenario, it would be necessary only to choose a set of routes and name the scenario.

The user would then be presented with the program main menu, shown in the bottom half of Figure 2. This menu would include options to manage the scenario description files, to change parameters used within the scheduling program, to schedule watches for a route within the current scenario, and to create or show reports.

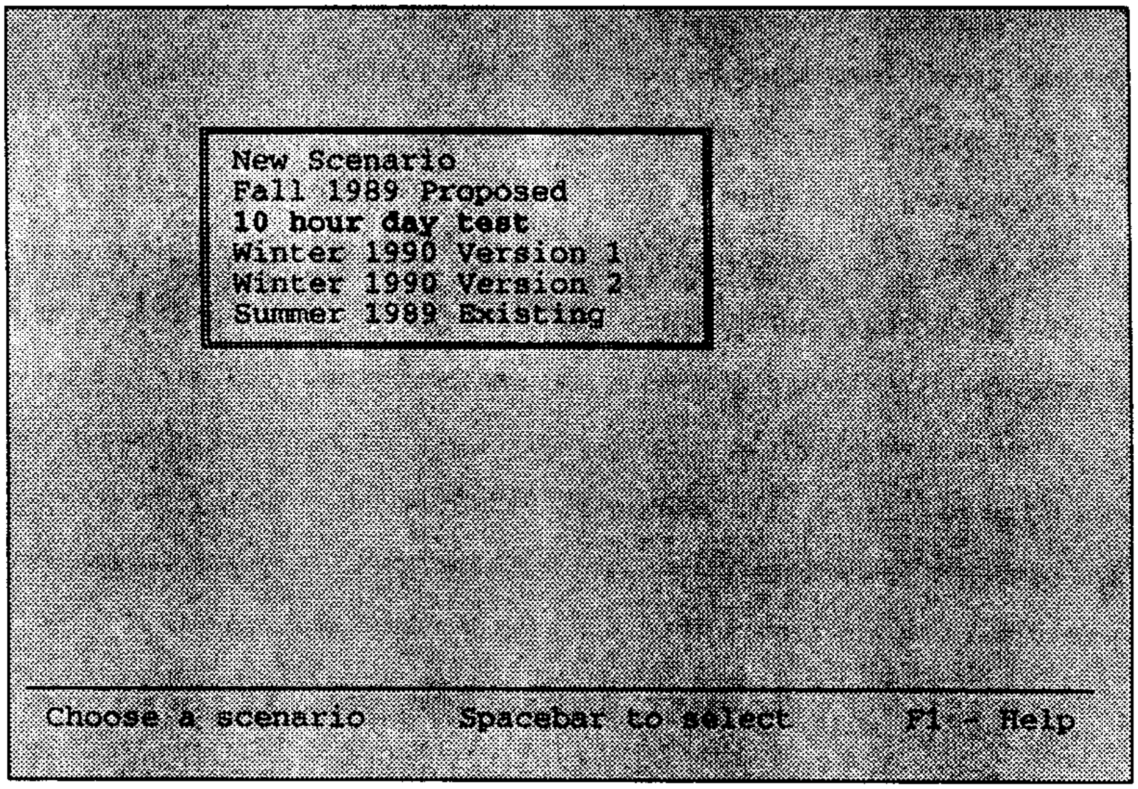
SCHEDULING PARAMETERS

At the systemwide (or scenario) level, several options could be altered for a given scenario. Figure 3 shows the Scenario Parameters menu. The vessels, vessel classes, terminals, vessel travel times, and other definitions would have to be changed on occasion, but these would most likely be changed with an external utility program (that is, outside of the watch scheduling program) so that the changes could be shared with the vessel scheduling software. Confusion and errors would result from the use of different values for these parameters in the vessel and watch scheduling systems.

For each scenario, a set of legal shift types would be maintained, including the existing default shift types, as well as any others that the user chose to define. Figure 4 shows how a shift type could be defined.

SCHEDULING PROCESS

Crews (watches) would be scheduled on one route at a time. The program could be configured to schedule watches for all crews or for deck crews only, for whom the schedule process is most difficult. If below deck crews and masters and mates were to be



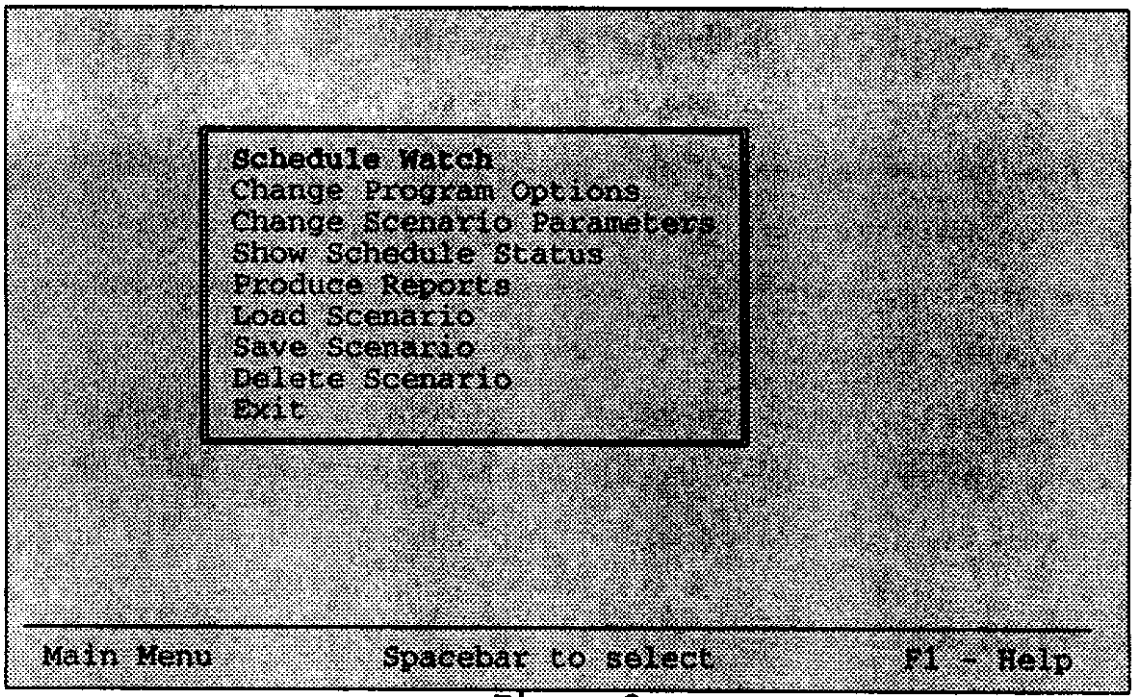


Figure 2
Opening Menu and Main Menu

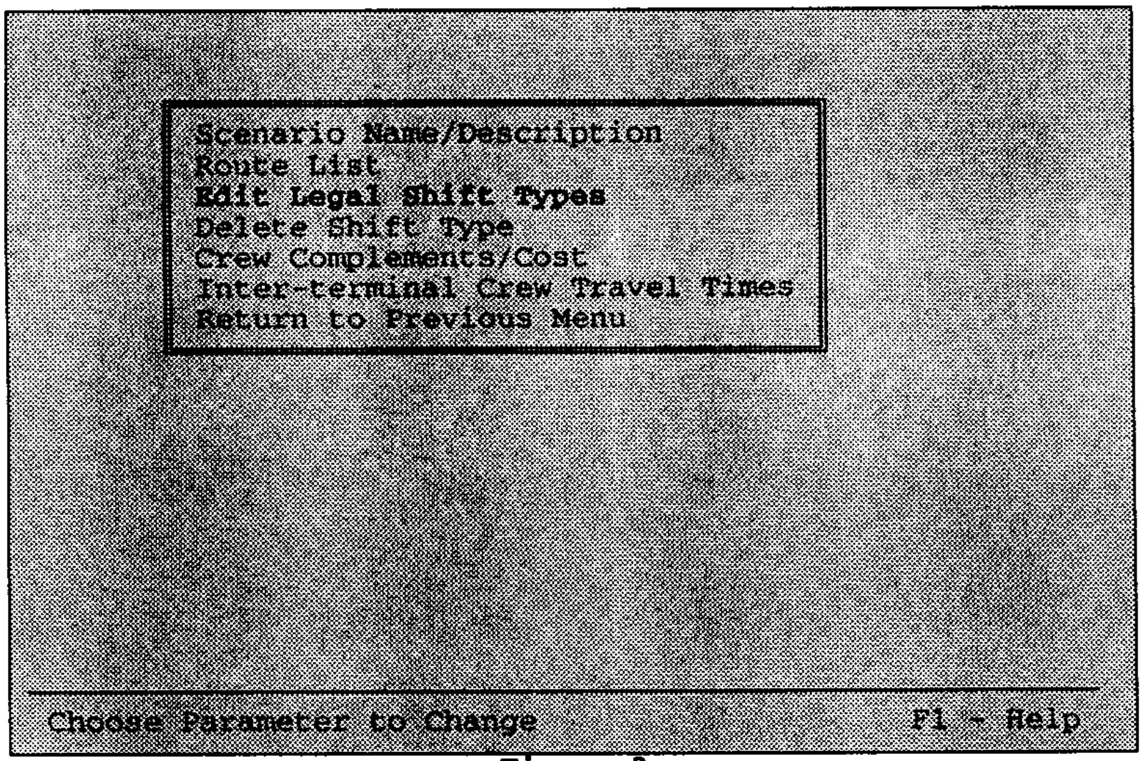
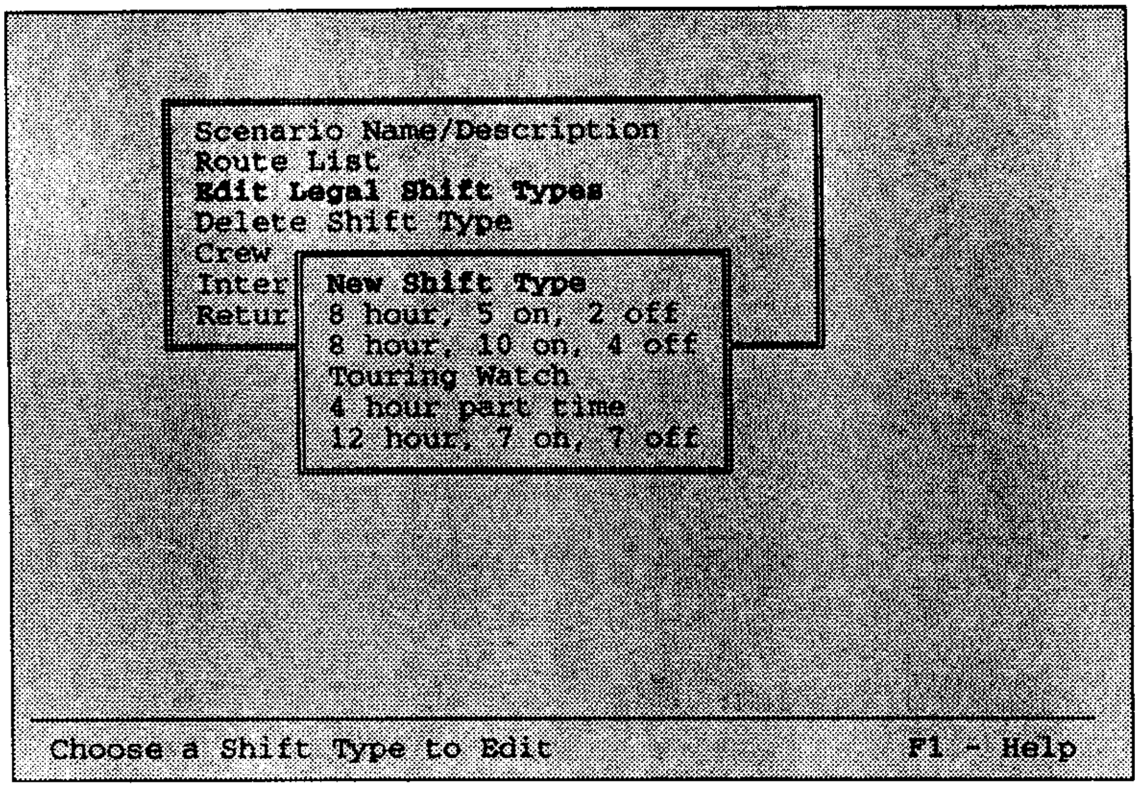


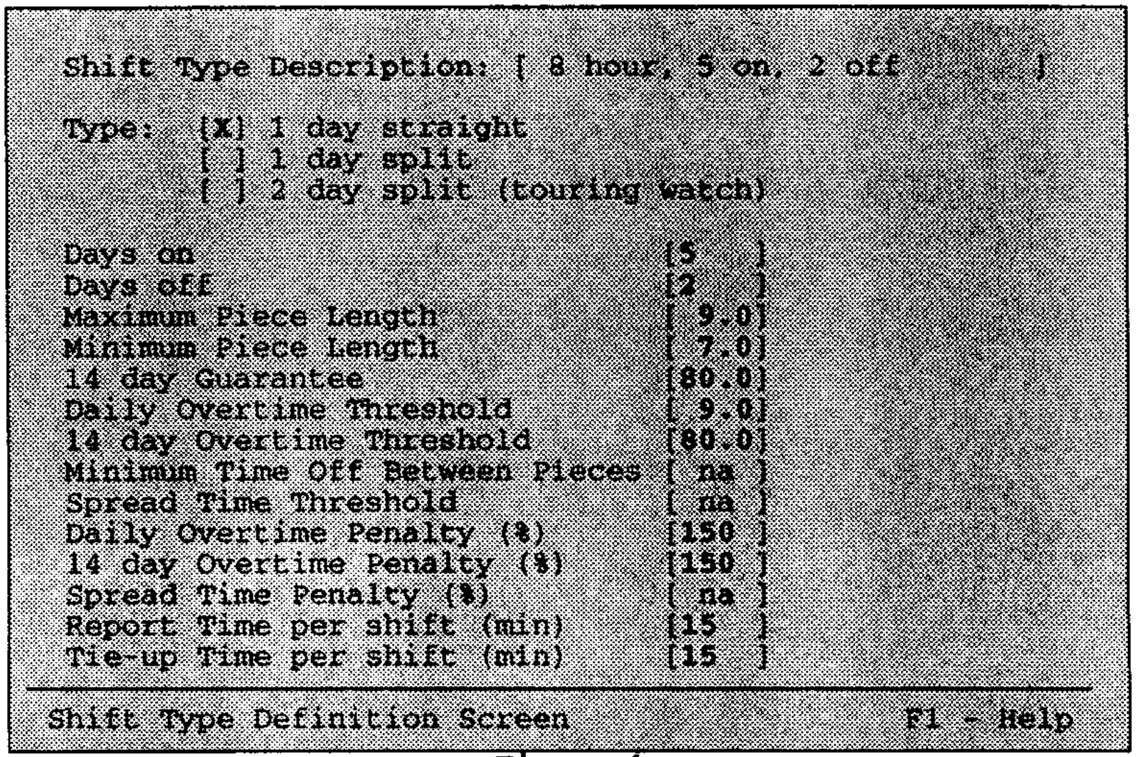
Figure 3
Scenario Parameters Menu



```
Scenario Name/Description
Route List
Edit Legal Shift Types
Delete Shift Type
Crew
Inter
Retur

New Shift Type
8 hour, 5 on, 2 off
8 hour, 10 on, 4 off
Touring Watch
4 hour part time
12 hour, 7 on, 7 off

Choose a Shift Type to Edit                    F1 - Help
```



```
Shift Type Description: [ 8 hour, 5 on, 2 off        ]

Type:  [X] 1 day straight
       [ ] 1 day split
       [ ] 2 day split (touring watch)

Days on                           [5   ]
Days off                          [2   ]
Maximum Piece Length              [ 9.0]
Minimum Piece Length              [ 7.0]
14 day Guarantee                  [80.0]
Daily Overtime Threshold          [ 9.0]
14 day Overtime Threshold         [80.0]
Minimum Time Off Between Pieces   [ na ]
Spread Time Threshold             [ na ]
Daily Overtime Penalty (%)        [150 ]
14 day Overtime Penalty (%)       [150 ]
Spread Time Penalty (%)           [ na ]
Report Time per shift (min)       [15  ]
Tie-up Time per shift (min)       [15  ]

Shift Type Definition Screen                   F1 - Help
```

Figure 4
Legal Shift Type Definition

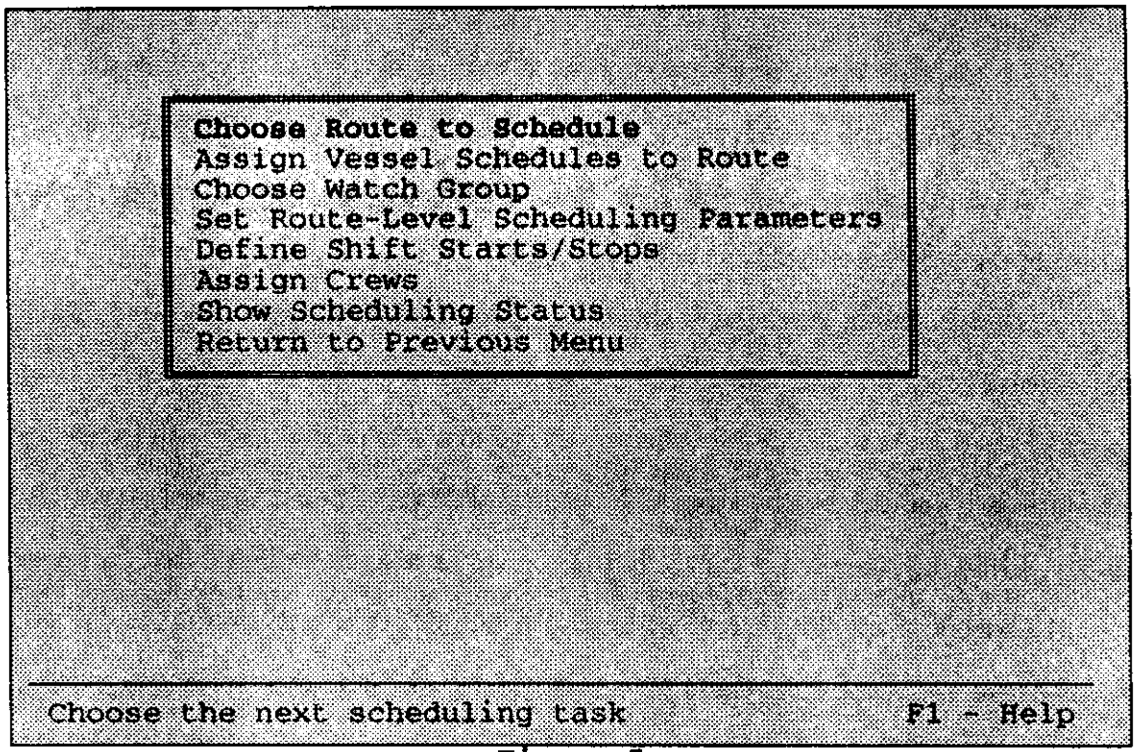
scheduled, each watch group would be scheduled independently. It might even be desirable to schedule two or more groups of deck workers on a vessel in order to allow a subset of a deck crew to be assigned to a smaller vessel for one day a week. (This would allow limited scheduling by position. If full scheduling by position was desired, this section of the program would have to be expanded.)

The Scheduling Submenu is shown in Figure 5. Once a route and watch group had been selected for scheduling, a four-step process would be needed to schedule a watch group from scratch. The first step would be to associate the route with the vessel schedules that defined the operating hours and possible relief points for each day in the 14-day period (to allow for special schedules on weekends and holidays, fuel runs, and special events). Secondly, the route-level scheduling parameters would be chosen, including the shift types that could be considered for the route and watch group being scheduled, the relief terminal for each vessel, and restrictions on the scheduling process. Thirdly, daily vessel schedules would be broken into work shifts. Finally, crews would be assigned to the work shifts.

Figure 6 shows the first two steps of this process. In the top half of Figure 5, a vessel schedule is assigned to each day of the two-week period. Choosing a "base schedule" would fill all 14 days with the same vessel schedule; then individual days could be selected if a different vessel schedule was desired for those days.

In the bottom half of Figure 6 is an example of some of the types of scheduling parameters one might wish to change before defining and assigning watch shifts. The restrictions checked would be used in determining where to suggest a shift break. The relief point determines where shifts can be broken — other time points would be ignored by the software.

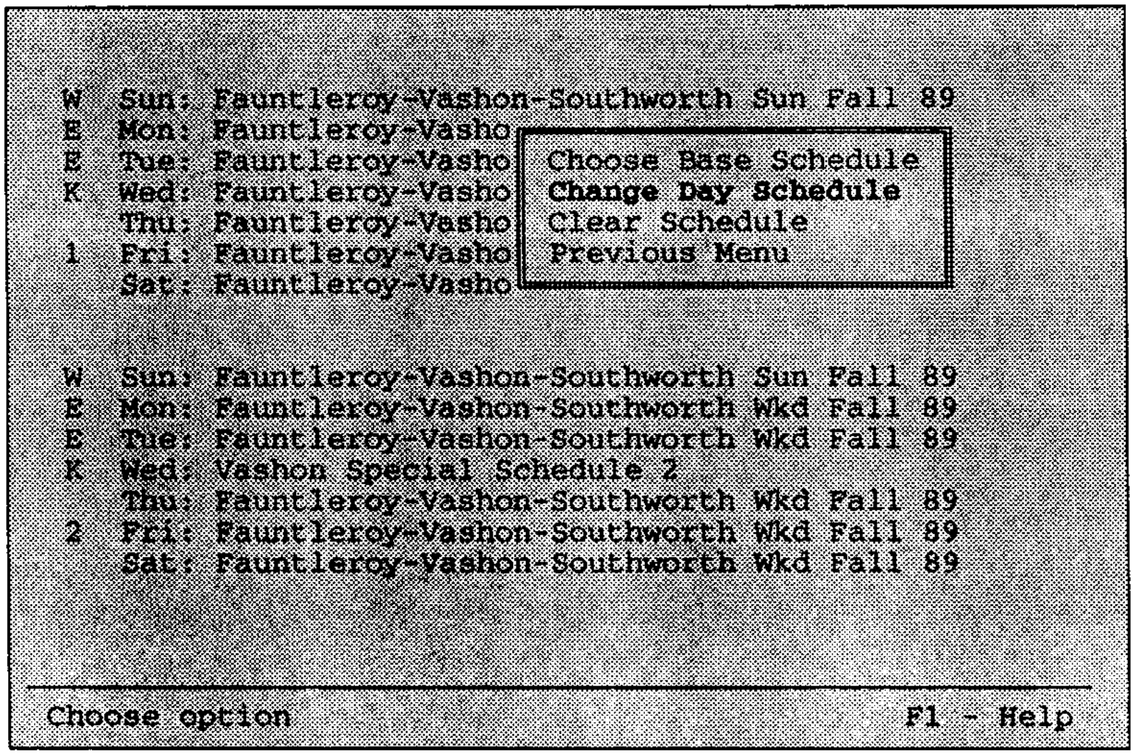
Figure 7 shows the screen that would be used to break a vessel schedule into work shifts. (Note that an entire week would be shown.) The scheduling program would use the restrictions defined for the route to determine the most likely breaking point between



Choose Route to Schedule
Assign Vessel Schedules to Route
Choose Watch Group
Set Route-Level Scheduling Parameters
Define Shift Starts/Stops
Assign Crews
Show Scheduling Status
Return to Previous Menu

Choose the next scheduling task F1 - Help

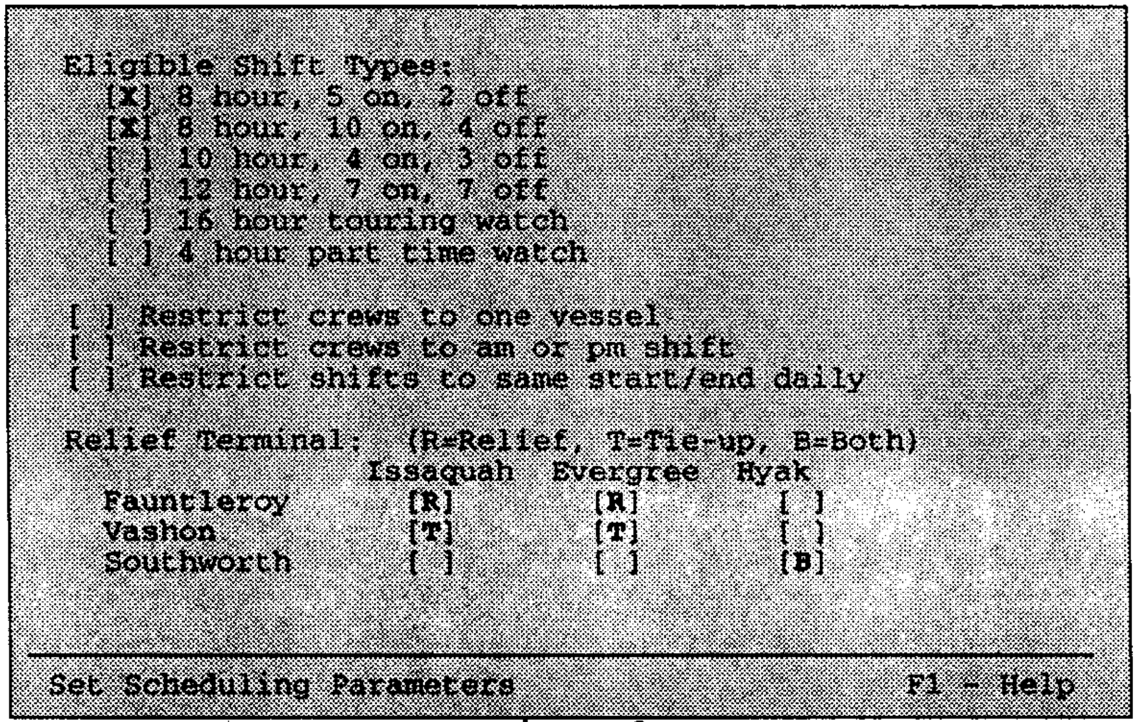
Figure 5
Scheduling Submenu



```
W  Sun: Fauntleroy-Vashon-Southworth Sun Fall 89
E  Mon: Fauntleroy-Vasho
E  Tue: Fauntleroy-Vasho   Choose Base Schedule
K  Wed: Fauntleroy-Vasho   Change Day Schedule
   Thu: Fauntleroy-Vasho   Clear Schedule
1  Fri: Fauntleroy-Vasho   Previous Menu
   Sat: Fauntleroy-Vasho

W  Sun: Fauntleroy-Vashon-Southworth Sun Fall 89
E  Mon: Fauntleroy-Vashon-Southworth Wkd Fall 89
E  Tue: Fauntleroy-Vashon-Southworth Wkd Fall 89
K  Wed: Vashon Special Schedule 2
   Thu: Fauntleroy-Vashon-Southworth Wkd Fall 89
2  Fri: Fauntleroy-Vashon-Southworth Wkd Fall 89
   Sat: Fauntleroy-Vashon-Southworth Wkd Fall 89

Choose option                                   F1 - Help
```



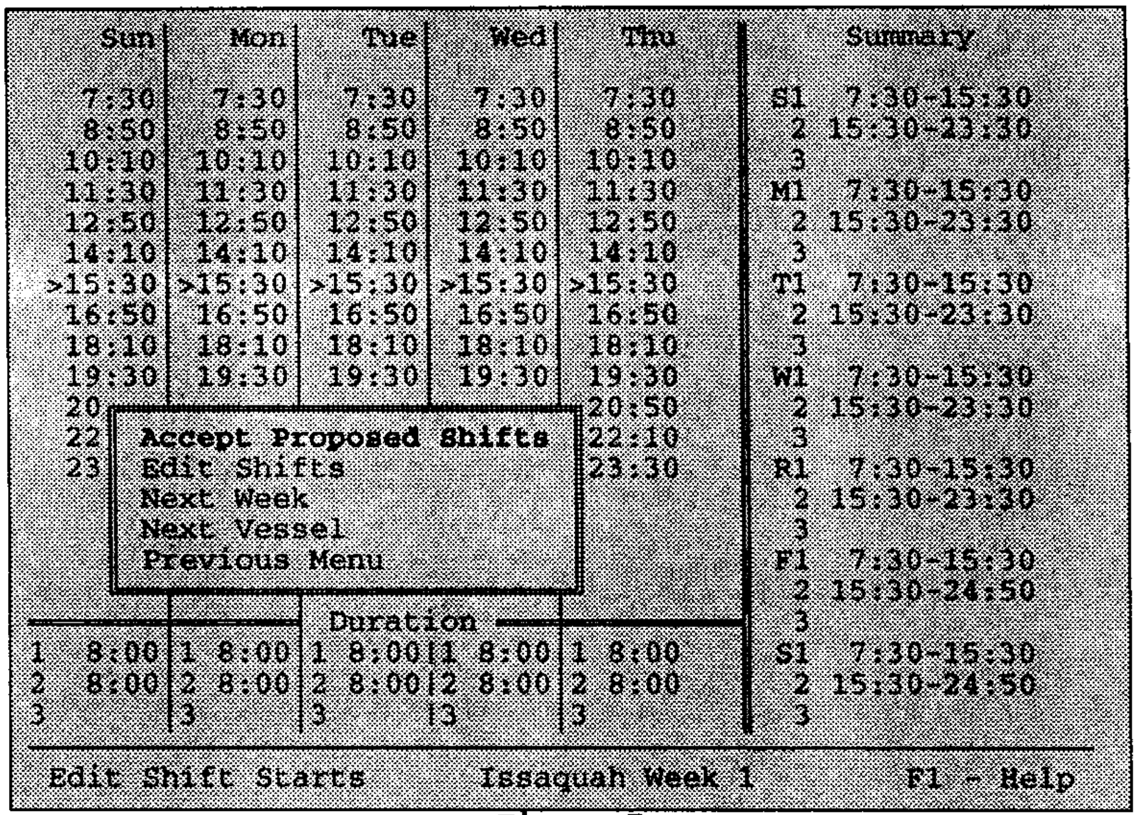
```
Eligible Shift Types:
  [X] 8 hour, 5 on, 2 off
  [X] 8 hour, 10 on, 4 off
  [ ] 10 hour, 4 on, 3 off
  [ ] 12 hour, 7 on, 7 off
  [ ] 16 hour touring watch
  [ ] 4 hour part time watch

[ ] Restrict crews to one vessel
[ ] Restrict crews to am or pm shift
[ ] Restrict shifts to same start/end daily

Relief Terminal:  (R=Relief, T=Tie-up, B=Both)
                Issaquah  Evergree  Hyak
  Fauntleroy       [R]       [R]     [ ]
  Vashon           [T]       [T]     [ ]
  Southworth       [ ]       [ ]     [B]

Set Scheduling Parameters                        F1 - Help
```

Figure 6
Assigning Vessel Schedules to Route and Setting Parameters



```
     Sun|    Mon|    Tue|    Wed|    Thu   ||      Summary

    7:30|   7:30|   7:30|   7:30|   7:30   || S1  7:30-15:30
    8:50|   8:50|   8:50|   8:50|   8:50   ||  2 15:30-23:30
   10:10|  10:10|  10:10|  10:10|  10:10   ||  3
   11:30|  11:30|  11:30|  11:30|  11:30   || M1  7:30-15:30
   12:50|  12:50|  12:50|  12:50|  12:50   ||  2 15:30-23:30
   14:10|  14:10|  14:10|  14:10|  14:10   ||  3
  >15:30| >15:30| >15:30| >15:30| >15:30   || T1  7:30-15:30
   16:50|  16:50|  16:50|  16:50|  16:50   ||  2 15:30-23:30
   18:10|  18:10|  18:10|  18:10|  18:10   ||  3
   19:30|  19:30|  19:30|  19:30|  19:30   || W1  7:30-15:30
   20|                             |20:50   ||  2 15:30-23:30
   22|  Accept Proposed Shifts     |22:10   ||  3
   23|  Edit Shifts                |23:30   || R1  7:30-15:30
     |  Next Week                  |        ||  2 15:30-23:30
     |  Next Vessel                |        ||  3
     |  Previous Menu              |        || F1  7:30-15:30
     |                             |        ||  2 15:30-24:50
  ----------------- Duration ---------------||  3
1  8:00|1 8:00|1 8:00|1 8:00|1 8:00        || S1  7:30-15:30
2  8:00|2 8:00|2 8:00|2 8:00|2 8:00        ||  2 15:30-24:50
3      |3     |3     |3     |3             ||  3
-----------------------------------------------------------
 Edit Shift Starts        Issaquah Week 1        F1 - Help
```

Figure 7
Defining Shifts

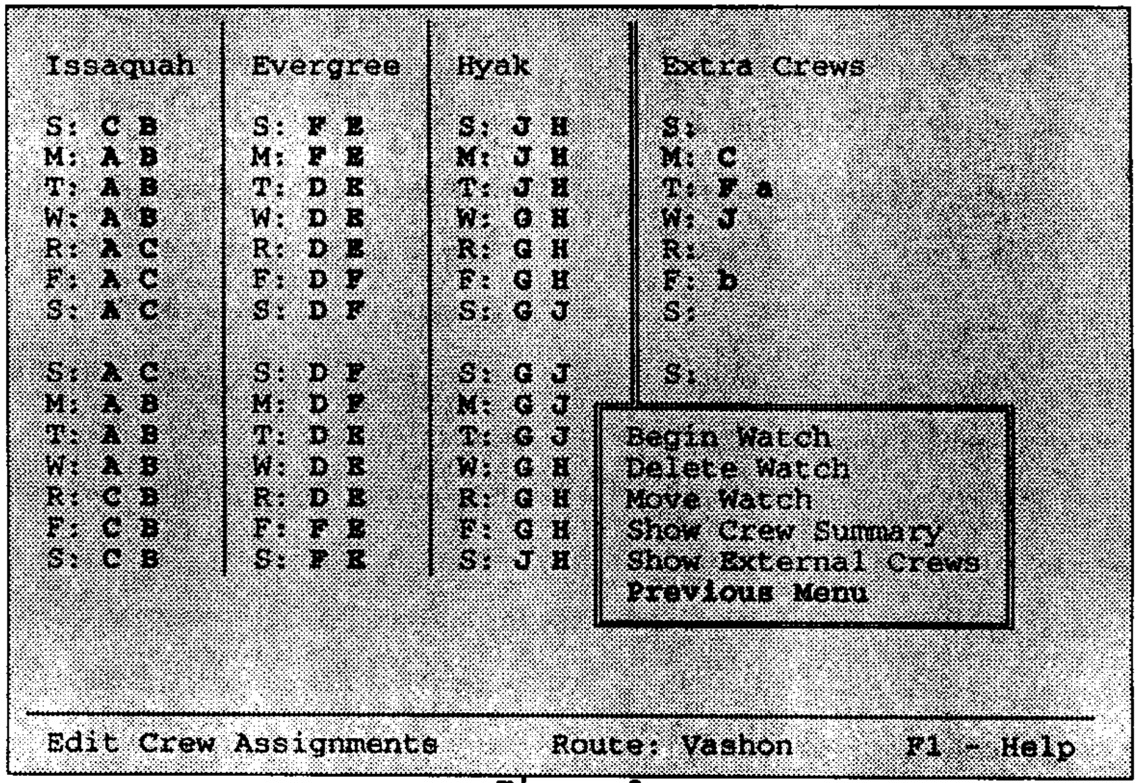
shifts. If several shift types were legal for the route being scheduled, the program would attempt to determine the legal shifts that would best fit a vessel schedule of a given duration. The WSF planner could override the computer's selection.

Finally, Figure 8 shows the screen that would be used to assign watches to shifts. To schedule a watch, the user would move the cursor to the point where a new watch should be started. Pressing the spacebar would bring up the menu shown in the figure. If the Begin Watch option was chosen, then the program would attempt to determine the appropriate shift type to schedule. If more than one shift type could be scheduled (e.g., if it was an eight-hour shift and either a five- or ten-day work week could be specified), then the user would be prompted to choose between the two. If the watch was begun in a morning shift, but a subsequent morning shift was already assigned to another crew, then the program would shift the new crew to the afternoon, to another vessel, or to the extra watch list, depending on the settings specified in the route scheduling parameters.

Extra crews (i.e., crews that must split their time between two or more vessels but have only been assigned to the first of the two) are shown on the right side of the screen. They include both crews that have been assigned locally to the route being scheduled, as well as crews that have been assigned on other routes and could be transported to the current route.

Instead of using the menu, the user could also type a letter to indicate the assignment desired. If the letter corresponded to a crew that had already been assigned, then that crew would be reassigned to the spot where the cursor was located, swapping with any crew already assigned to the shift. If the letter did not correspond to an existing crew, then a new watch would be created.

Throughout this process, the program would keep track of the crews that were eligible for each shift, indicating shift types by color coding. If the user attempted to assign a crew to a shift where it is not eligible, the program would note the error and allow the user to override it.



```
Issaquah   Evergree   Hyak      Extra Crews

S: C B     S: F E     S: J H    S:
M: A B     M: F E     M: J H    M: C
T: A B     T: D E     T: J H    T: F a
W: A B     W: D E     W: G H    W: J
R: A C     R: D E     R: G H    R:
F: A C     F: D F     F: G H    F: b
S: A C     S: D F     S: G J    S:

S: A C     S: D F     S: G J    S:
M: A B     M: D F     M: G J
T: A B     T: D E     T: G J   Begin Watch
W: A B     W: D E     W: G H   Delete Watch
R: C B     R: D E     R: G H   Move Watch
F: C B     F: F E     F: G H   Show Crew Summary
S: C B     S: F E     S: J H   Show External Crews
                               Previous Menu

Edit Crew Assignments        Route: Vashon        F1 - Help
```

Figure 8
Defining Crew Assignments

Ideally, the crew scheduling program would attempt to find an optimum schedule for each watch and propose it. This capability would allow the prototype design to approach the "Specialized Computer Algorithm" alternative described in Chapter 3. Unfortunately, this capability will be very difficult to add, and has not been included in the current prototype design.

CONCLUSIONS AND RECOMMENDATIONS

This chapter discusses the conclusions and recommendations the project team drew from the development of the prototype described in the previous chapter.

CONCLUSIONS

The prototype development effort resulted in the following conclusions.

- No computer algorithms exist in the literature that make the proposed crew scheduling system less costly to build than previously estimated (roughly $200,000).
- The prototype development indicated that the previous cost estimate is actually optimistic, because creating a user interface that would eliminate or reduce the complexity of the scheduling process appears to be a very difficult task. Creating the user interface would be a more difficult task than programing the algorithms to actually compute the watch schedules.
- Unless this complexity could be eliminated, the system would require several days of training to use, and if the system was not used routinely, refresher training would be necessary before the system was used again.
- A smaller system that performed specific parts of the scheduling analysis (but not all facets of the analysis) might be more useful to the WSF staff than a full computer system, simply because the smaller system could take the complexity out of the computer program, making it easier to learn and operate.

RECOMMENDATIONS

The project team makes the following recommendations to WSF regarding the development of a computer aid for the watch scheduling process.

- The WSF should not pursue further development of a computerized system to aid in watch (crew) scheduling at this time.
- Further development of a computerized system is recommended only if at least one of the following three conditions occur:
  - WSF decides to schedule crew members by position rather than by watch,
  - WSF labor rules are changed to the extent that the current schedules are no longer valid, or
  - wholesale changes in the vessel schedules (all routes) are expected to occur on a routine basis.
- A more effective use of the WSF's resources would be to hire a consultant to examine the impacts of alternative work rules on the service that WSF can provide.

DISCUSSION OF RECOMMENDATIONS

Hiring a consultant would provide answers to the basic questions WSF has regarding watch scheduling options. This would meet WSF's short-term needs and provide a practical benefit. A consultant study could also be done for less money than the development of the computer aided scheduler, and the consultant study would have much less risk associated with it than the software development effort. (It is possible that the scheduling software would not meet the WSF's needs even after the expected $200,000 expenditure.)

A consulting contract would provide answers to the basic WSF questions regarding the continuing labor negotiations. It would also provide answers to WSF planning personnel about the labor concessions that would be most valuable in terms of their ability to provide improved service, and the concessions the WSF could afford to give to the labor unions to gain the desired service improvements.

In the long term, a computer system would provide the WSF with more flexibility than a one time consulting effort, but given the stable history of WSF vessel schedules, it is not clear that the WSF would use the computer system often enough to warrant the additional cost associated with developing and then operating that system. Further, vessel schedule stability means that for most of the WSF watch scheduling needs a computer is not necessary.

Answers from a consultant project could be obtained much more quickly than if a computer system was first developed and then applied by WSF staff. Computer system development would take at least two years. A consultant's study could be completed in nine months to one year.

Finally, WSF might also stipulate that any tools developed by the consultant as part of the proposed study should be given to WSF as part of the project. This might result in some simple tools for use by WSF planners that would aid in future analyses, while at the same time providing the WSF planners with the end results that were the primary purpose of the scheduler development. Even the consultant did not develop any special tools, the WSF would still have the project results to use when negotiating alternative labor rules and the impacts of those rules on possible route structures and schedules.

APPENDIX A

PROGRAMMING INSTRUCTIONS

APPENDIX A
PROGRAMING INSTRUCTIONS

CHANGE PARAMETERS

Each of these menu selections changes a table or set of tables that are used within the scheduling software. Return from any menu item in this list is to this menu. Return from this menu itself (i.e., the ESC key or the "Return" menu selection) takes the user back to the Main Menu.

Scenario Name/Description

This allows the user to change the file name associated with the current crew schedule. It also lets the user create or edit a text description of the schedule being analyzed.

When the user has completed this item, control is returned to the Change Parameters menu.

Route List

To make the best use of crews, some crews will be scheduled to operate vessels on more than one route (i.e., Seattle-Bremerton and Kingston-Edmonds). However, the user might want to exclude some routes from consideration in the crew scheduling effort (it could cut down execution time for the program.) The route list function should show all of the routes in the system (and allow the addition of new routes) and let the user identify which of the routes should be included in the scenario being analyzed. Once a route has been selected for inclusion, the program should offer a second menu which allows the user to identify the specific datafiles which contains the vessel schedules for each day for the two week period being scheduled. (See the top half of Figure 6 in the main text.)

The program flow would be from the menu which listed possible routes (some flagged as included, and others flagged as not included) to a second menu which listed the datafiles for a highlighted route. From this datafile list, control should go back to the route menu, and when the user is done with the route menu, program control should return to the Change Parameters menu. Note that you can get to these same two menus from the Schedule Watch option of the Main Menu.

Edit Legal Shift Types

This allows the user to create or edit tables which control the rules which are used to define when a crew's shifts begin and end. It will probably be a two menu process. The first menu allows for selecting a specific shift type. The second menu is for editing (or creating) the table which contains the parameters associated with that shift type. The bottom of Figure 4 gives a good example of how the two menu process should work.

Selecting Edit Legal Shift Type should bring up the list of existing legal shift types. Selection of the shift type should bring up the table shown at the bottom of Figure 4. When the user is done editing this table (or if they hit ESC) control should be returned to the Change Parameters menu.

Delete Shift Type

This menu item would allow the user to delete specific types of shifts from the legal shift listing. (Provide a list of the legal shifts and a method for marking what should be deleted. The list could look like the window insert in Figure 4. After deletion, allow deletion of additional shift types.)

When the user has completed deleting shift types, control is returned to the Change Parameters menu.

Crew Complements/Cost

The scheduler assigns entire crews to vessels not individuals. Each type of vessel requires a different number of people in its crew. This menu should list each type of vessel, the number of crew members of each kind and their basic hourly pay rates.

When the user has completed editing this table, control is returned to the Change Parameters menu.

Inter-terminal Crew Travel Times

Crews that serve on routes away from their normal terminal get paid extra money for their travel. (That is, a crew which normally works the Seattle-Bremerton route gets travel payments if they are assigned to work the Clinton-Mukilteo route.) This menu item should bring up a table which lists the travel time and pay associated with traveling (by car) from each terminal to every other terminal.

When the user has completed this item, control is returned to the Change Parameters menu.

Return

Takes the user back to the Main Menu.

SCHEDULE WATCH

This selection from the main menu starts the major function of the scheduling system. Most of the items under the Schedule Watch Menu (Figure 5 in the main body of this report) contain additional sub-menus.

Choose Route To Schedule

The user will use this menu to select which route s/he will work on next. The menu should present them with a list of the selected routes chosen for this scenario. (This list or table comes from Route List created under Change Parameters above.) This menu will also let the user add new routes to the list under this menu item. (That is, this would be a second way to access the procedures under Route List.) However, the result from this menu item should be an "active" route, which will be used by the system for the crew scheduling process.

When completed with the above functions, program control should return to the Schedule Watch Menu. (Figure 5 in the main body of the paper.)

Assign Vessel Schedules to Route

This is the second half of the route selection process described under Route List description above. (That is, the user is given the chance to identify or change the specific datafiles that contain the schedule information for all 14 days in the two week schedule period. The difference is that from this starting point, only the active route (see the previous section, "Choose Route Selection" to define "active route") can have the datafiles identified or changed.

Program control from this process should return to the Schedule Watch Menu. (Figure 5 in the main body of the paper.)

Choose Watch Group

The different groups of WSF crew members that work on a vessel are called "watches." Each vessel has three "watches" per shift, one for members of the Masters, Mates and Pilots Union, one for Inland Boatmen's Union personnel, and one for the Marine Engineers Beneficial Association personnel. The three unions use different rules for crewing boats.

This step is simply choosing between the three unions. That is, does the user want to schedule masters, engineers or seamen right now?

When the union to be scheduled has been selected, program control from should return to the Schedule Watch Menu. (Figure 5 in the main body of the paper.)

(Set) Route Level Scheduling Parameters

This is the figure shown at the bottom of Figure 6. The parameters included in the figure control how the computer program will try to schedule the active route. This is an informational table rather than an editing table. It should display the active parameters that have been set using the other menu options under the "Schedule Watch" heading.

Define Shift Starts/Stops

For all three unions, the current work rules state that all shifts have to start and stop at specific terminals. (i.e., all engineers have to get on and off the boat in Seattle for all

Seattle-Bremerton and Seattle-Winslow runs.) Thus, to legally start or end a shift, the vessel must be docked at one of those terminals. This menu item would allow the user to add or change the terminals at which crews may start and end their work shifts for each of the legal routes. (See the routes table under "Route List" under "Change Parameters.")

When done with these changes, program control should return to the Schedule Watch Menu. (Figure 5 in the main body of the paper.)

Assign Crews

This is the difficult part of the program. It entails several different screens, starting with the main screen in Figure 7, then adding the window insert shown on Figure 7 and then progressing to Figure 8.

Show Scheduling Status

This is a summary table which shows which routes have crews assigned to all trips, which routes still do not have crew assignments, which crews have been assigned to some trips, but still do not have complete 40 hour/week work assignments, and when (days) those crews are legally available for work.

Program control from this process should return to the Schedule Watch Menu. (Figure 5 in the main body of the paper.)

Return To Previous Menu

Takes the user back to the Main Menu.

APPENDIX B

PROGRAM CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "crew_aux.c"

#define NUM_VESSEL    6
#define NUM_TERMINAL 20

int start_tm[10], dep_time[10][20];
int travel_tm[300], from[300], to[300], ves[300];
int num_tt, dept_tm[NUM_VESSEL][20], bk[7][20];
int peak_load[25], off_peak_load[25], peak_tm[25], off_peak_tm[25];

int n13[NUM_VESSEL];
int num_vessel;
int num_terminal;
int x1st_break_pt[NUM_VESSEL][7], x2nd_break_pt[NUM_VESSEL][7];

void watch15_basic(int,int);
void watch15_OK(void);
void watch15_ED(int);
void read_sched(int);
void load_stimes(int);
void load_ttimes(int);
void load_htimes(int);
void load_pktimes(void);
void load_ltimes(void);
void show_summary(int);
void display_time(int);

main()      /* define a shift */
    {
    char answer;
    int i, j, ii = 1, jj = 0;

    load_pktimes();
    load_ltimes();
    read_sched(0);
    load_stimes(0);
    load_ttimes(0);
    load_htimes(0);
    cleanscreen;
    SET_navy;
    poscur(5,10);
    printf("This is a demo program to show the option 5 is possible ");
    SET_green;
    poscur(7,15);
    printf("further work need to be done:");
    poscur(8,20);
    printf("extend break_points array into second week");
    poscur(9,20);
    printf("small bug in cursor position");
    poscur(10,20);
    printf("combine with crew watch schedule program");
    enter(14);
    for ( i = 0; i < 7; i++)
        {
        for ( j = 0; j < NUM_VESSEL; j++)
            {
            x1st_break_pt[j][i] = 1;
            x2nd_break_pt[j][i] = -2;
            }
        for ( j = 0; j < 20; j++)
            bk[i][j] = 0;
        bk[i][1] = 0;
        }
  begin:
      watch15_basic(ii,jj);
      SET_navy;
      poscur(24,1);
      printf(" (1) Accept Proposed Shift  (2) Edit Shifts ");
      if (ii == 1)
          printf(" (3) Next Week\n");
      else
          printf(" (3) Previous Week\n");
      printf(" (4) Next Vessel   (5) Previous Menu");
   input:
      SET_green;        choose(25,40);        poscur(25,60);
      COLOR_off;        answer=getch();       putch(answer);
      switch(answer-48)
```

```
        {
        case 1 : watch15_OK();      break;
        case 2 : watch15_ED(jj);    break;
        case 3 : ii = ii % 2 + 1;   break;
        case 4 : jj = (jj+1) % 2;   ii = 1;     break;
        case 5 : printf("\n\n");    exit(0);
        default: c_error(24,answer);    COLOR_off;      goto input;
        }
    goto begin;
    }

#define ENTER 13
#define ESC   27
#define SPACE 32
#define UP    72
#define LEFT  75
#define RIGHT 77
#define DOWN  80

void watch15_ED(int jj)
    {
    int row, col, i, j, k, hh, mm, xx, press_enter[7];
    int b1, b2, d1, d2, s1, s2;
    char key, ch;

    SET_navy;
    poscur(24,1);
    printf(" Use Arrow Key to Select the Break Point; Use Enter Key to Pick Up");
    poscur(25,1);
    printf(" the Break Point; and Use the SpaceBar to Return to Previous Menu ");
    k = 0;
    row = 3;
    col = 3 + 8 * k;
    for ( i = 0; i < 7; press_enter[i] = 0, i++);
    b1 = 1;  b2 = -2;
begin:
    /* Summary */
    show_summary(jj);
    /* ------------- input key ----------------------- */
    poscur(row,col);
    ch = getch();
    if (ch == SPACE) goto end;
    if (ch == LEFT || ch == RIGHT)
        {
        if (bk[k][b1] == 0) printf(" ");
        if (ch == RIGHT) k++;
        if (ch == LEFT)  k--;
        if (k < 0) k = 6;
        if (k > 6) k = 0;
        if (press_enter[k] == 0)
            {
            poscur(3,3+8*k);
            printf(" ");
            }
        }
     if (ch == ENTER && press_enter[k] == 0)
        {
        x1st_break_pt[jj][k] = b1;
        bk[k][b1] = 1;
        press_enter[k]++;
        b2 = b1;
        goto show_break_pt;
        }
     if (ch == ENTER && press_enter[k] == 1)
        {
        x2nd_break_pt[jj][k] = b1;
        bk[k][b1] = 2;
        press_enter[k]++;
        goto show_break_pt;
        }
     if (ch == UP || ch == DOWN)
        {
        if (bk[k][b1] == 0) printf(" ");
        if (ch == UP) b1--;
        if (ch == DOWN) b1++;
        if (press_enter[k] == 0)
            {
            if (b1 < 1 )  b1 = 1;
            if (b1 > n13[jj]-1)  b1 = n13[jj]-1;
            }
```

```
          if (press_enter[k] == 1)
              {
              if (b2 < 1) b2 = 1;
              if (b1 < b2) b1 = b2+1;
              if (b1 > n13[k]-1) b1 = n13[k]-1;
              }
          }
show_break_pt:
     row = 2 + b1;
     col = 3 + k * 8;
     poscur(row,col);
     if (bk[k][b1] == 0)
         {
         SET_navy;
         printf("+");
         }
     if (bk[k][b1] == 1)
         {
         SET_green;
         printf("*");
         }
     if (bk[k][b1] == 2)
         {
         SET_yellow;
         printf("*");
         }
     /*
     SET_red;
     for ( i = 0; i < 7; i++)
         {
         j = 3 + 8 * i;
         poscur(22,j);
         printf(" %3d  ",n13[jj]);
         poscur(23,j);
         printf(" %3d  ",x1st_break_pt[jj][i]);
         poscur(24,j);
         printf(" %3d  ",x2nd_break_pt[jj][i]);
         }
     poscur(25,5);
     printf("  k%3d  b1%3d  b2%3d   ", k, b1, b2);
     */
     for ( i = 0; i < 7; i++)
         for ( j = 0; j < 20; j++)
             {
             if (bk[i][j] == 1) x1st_break_pt[jj][i] = j;
             if (bk[i][j] == 2) x2nd_break_pt[jj][i] = j;
             }
     goto begin;
  end:
      return;
      }
  void show_summary(int jj)
      {
      int i, j, k, s1, s2, d1, d2, xx;
      SET_white;
      for ( i = 0; i < 7; i++)
          {
          s1 = x1st_break_pt[jj][i];
          s2 = x2nd_break_pt[jj][i];
          poscur(2+i*3,64);
          xx = dept_tm[jj][0];
          display_time(xx);
          printf("-");
          xx = dept_tm[jj][s1];
          display_time(xx);
          if (x2nd_break_pt[jj][i] <= 0)
              {
              poscur(3+i*3,64);
              xx = dept_tm[jj][s1];
              display_time(xx);
              printf("-");
              xx = dept_tm[jj][n13[jj]-1];
              display_time(xx);
              }
          else
              {
              poscur(3+i*3,64);
              xx = dept_tm[jj][s1];
              display_time(xx);
              printf("-");
```

```
            xx = dept_tm[jj][s2];
            display_time(xx);
            poscur(4+i*3,64);
            xx = dept_tm[jj][s2];
            display_time(xx);
            printf("-");
            xx = dept_tm[jj][n13[jj]-1];
            display_time(xx);
            }
        }
    /* duration */
    SET_cyan;
    for ( j = 0; j < 7; j++)
        {
        d1 = x1st_break_pt[jj][j];
        d2 = x2nd_break_pt[jj][j];
        poscur(20,4+8*j);
        xx = dept_tm[jj][d1]-dept_tm[jj][0];
        display_time(xx);
        if (x2nd_break_pt[jj][j] <= 0)
            {
            poscur(21,4+8*j);
            xx = dept_tm[jj][n13[jj]-1]-dept_tm[jj][d1];
            display_time(xx);
            }
        else
            {
            poscur(21,4+8*j);
            xx = dept_tm[jj][d2]-dept_tm[jj][d1];
            display_time(xx);
            poscur(22,4+8*j);
            xx = dept_tm[jj][n13[jj]-1]-dept_tm[jj][d2];
            display_time(xx);
            }
        }
    return;
    }
void display_time(int xx)
    {
    int hh, mm;
    hh = xx / 60;
    mm = xx % 60;
    if (mm < 10)
        printf("%2d:0%1d", hh, mm);
    else
        printf("%2d:%2d", hh, mm);
    return;
    }

void watch15_basic(int ii, int jj)
    {
    int i, j, hh, mm, next_day, xx;
    static char vessel[3][9] = { "Cathlamet", "Hiyu" };

    COLOR_off;
    cleanscreen;
    SET_yellow;
    subscrn("watch15.mnu");
    COLOR_off;
    set_screen(37,44);
    poscur(17,15);  printf(" %-9.9s ",vessel[jj]);
    set_screen(37,42);
    poscur(17,48);      printf(" Week %d ",ii);
    COLOR_off;
    /* Summary */
    SET_white;
    for ( i = 0; i < 7; i++)
        {
        poscur(2+i*3,64);
        xx = dept_tm[jj][0];
        display_time(xx);
        printf("-");
        xx = dept_tm[jj][x1st_break_pt[jj][i]];
        display_time(xx);
        if (x2nd_break_pt[jj][i] <= 0)
            {
            poscur(3+i*3,64);
            xx = dept_tm[jj][x1st_break_pt[jj][i]];
            display_time(xx);
            printf("-");
```

```
          xx = dept_tm[jj][n13[jj]-1];
          display_time(xx);
          }
      else
          {
          poscur(3+i*3,64);
          xx = dept_tm[jj][x1st_break_pt[jj][i]];
          display_time(xx);
          printf("-");
          xx = dept_tm[jj][x2nd_break_pt[jj][i]];
          display_time(xx);
          poscur(4+i*3,64);
          xx = dept_tm[jj][x2nd_break_pt[jj][i]];
          display_time(xx);
          printf("-");
          xx = dept_tm[jj][n13[jj]-1];
          display_time(xx);
          }
      }
  /* main schedule table */
  for ( i = 0; i < n13[jj]; i++)
      for ( j = 0; j < 7; j++)
          {
          poscur(2+i,4+8*j);
          xx = dept_tm[jj][i];
          hh = xx / 60;
          mm = xx % 60;
          next_day = 0;
          if ( hh >= 24 )
              {
              next_day = 1;
              hh -= 24;
              xx -= 24;
              }
          if ( i == x1st_break_pt[jj][j] || i == x2nd_break_pt[jj][j] )
              {
              SET_green;
              poscur(2+i,3+8*j);
              printf("*");
              }
          SET_white;
          if (next_day) COLOR_off;
          display_time(xx);
          }
   /* duration */
   SET_cyan;
   for ( j = 0; j < 7; j++)
       {
       poscur(20,4+8*j);
       xx = dept_tm[jj][x1st_break_pt[jj][j]]-dept_tm[jj][0];
       display_time(xx);
       if (x2nd_break_pt[jj][j] <= 0)
           {
           poscur(21,4+8*j);
           xx = dept_tm[jj][n13[jj]-1]-dept_tm[jj][x1st_break_pt[jj][j]];
           display_time(xx);
           }
       else
           {
           poscur(21,4+8*j);
           xx = dept_tm[jj][x2nd_break_pt[jj][j]]-dept_tm[jj][x1st_break_pt[jj][j]];
           display_time(xx);
           poscur(22,4+8*j);
           xx = dept_tm[jj][n13[jj]-1]-dept_tm[jj][x2nd_break_pt[jj][j]];
           display_time(xx);
           }
       }
    return;
    }

void watch15_OK()
    {
    cleanscreen;
    poscur(12,30);
    set_txt_blnk(35);   printf("Accept Proposed Shifts");
    COLOR_off;
    enter(20);
    poscur(20,40);
    printf("                              ");
    poscur(15,30);
    printf("please wait....");
    delay(2500);
    return;
    }

void read_sched(int m)
    {
    int i, j, k, l;
    char a[5];
    int terminal_num[20], vessel_num[20];
    FILE *fp, *fopen();
    static char name[15] = "sched000.dat";

    if (m >= 9)
        {
        m -= 10;
        name[6] = 49;
        }
    name[7] = 49+m;
    /* printf("%-15.15s\n",name); */
    if ((fp = fopen(name,"r"))==NULL)
        {
        file_msg(name);  exit(0);
        }
    fgets(a, 3, fp);
    l = atoi(a);
    for ( j = 0; j < l; j++)
        {
        if (j == 5) while((fgetc(fp))==13);
        fgets(a, 3, fp);
        terminal_num[j] = atoi(a);
        }
    while((fgetc(fp))==13);
    fgets(a, 3, fp);
    k = atoi(a);
    for ( j = 0; j < k; j++)
        {
        if (j == 5) while((fgetc(fp))==13);
        fgets(a, 3, fp);
        vessel_num[j] = atoi(a);
        }
    fclose(fp);
    /* printf("%5d",l);
    for ( i = 0; i < l; i++)
        {
        printf(" %3d",terminal_num[i]);
        }
    printf("\n");
    printf("%5d",k);
    for ( i = 0; i < k; i++)
        {
        printf(" %3d",vessel_num[i]);
        }
    printf("\n"); */
    return;
    }

void load_stimes(int jj)
    {
    int i, j, l, m, n, hh, mm;
    static char stime[12] = "stimes.000";
    char htimes[12], ttimes[12];
    FILE *fp, *fopen();

    if (jj >= 9)
        {
        jj -= 10;
        stime[8] = 49;
        }
    stime[9] = 49+jj;
    if ((fp = fopen(stime,"r"))== NULL)
        {
        file_msg(stime);   return;
        }
    fscanf(fp,"%d", &num_vessel);
    for ( i = 0; i < num_vessel; i++)
        {
        start_tm[i] = 0;
        fscanf(fp,"%s",ttimes);
        l = strlen(ttimes);
        hh = 0;
        mm = 0;
        for ( m = 0; m < l; m++)
            {
            n = m;
            if (ttimes[m] == 58)  break;
            hh=hh*10+ttimes[m]-48;
            }
        for ( m = n+1; m < l; m++)
            mm=mm*10+ttimes[m]-48;
        start_tm[i] = 60 * hh + mm;
        fscanf(fp,"%s",htimes);
        if (htimes[0] == 80 || htimes[0] == 112)
            start_tm[i] += 720;
        }
    fclose(fp);
    return;
    }

void load_ttimes(int jj)
    {
    int i, j, k, l, m, n, hh, mm;
    static char ttime[12] = "ttimes.000";
    FILE *fp, *fopen();

    if (jj >= 9)
        {
        jj -= 10;
        ttime[8] = 49;
        }
    ttime[9] = 49+jj;
    if ((fp = fopen(ttime,"r"))== NULL)
        {
        file_msg(ttime);   return;
        }
    l = 0;
    fscanf(fp,"%d %d", &num_terminal, &num_vessel);
    for ( i = 0; i < num_terminal; i++)
        for ( j = 0; j < num_terminal-1; j++)
            for ( k = 0; k < num_vessel; k++)
                {
                fscanf(fp,"%d %d %d %d", &from[l],&to[l],&ves[l],&travel_tm[l]);
                l++;
                }
    fclose(fp);
    num_tt = l;
    return;
    }

void load_htimes(int jj)
    {
    int i, j, k, l, m, n, num_stops, pre, start, next_day;
    int vessel, stop, hold, default_hold;
    int hh, mm, xx;
    int default_load;
    float speed;
    static char htime[12] = "htimes.000";
    FILE *fp, *fopen();

    if (jj >= 9)
        {
        jj -= 10;
        htime[8] = 49;
        }
    htime[9] = 49+jj;
    if ((fp = fopen(htime,"r"))== NULL)
        {
        file_msg(htime);   return;
        }
    fscanf(fp,"%d", &num_vessel);
    for ( i = 0; i < num_vessel; i++)
        {
        l = -1;
        n13[i] = 0;
        xx = start_tm[i];
        dept_tm[i][n13[i]] = xx;
        n13[i]++;
        fscanf(fp,"%d",&num_stops);
        /*
        SET_red;
        printf("--------------------------------------------- %2d\n",num_stops);
        */
        next_day = 0;
        for ( j = 0; j < num_stops; j++)
            {
            SET_white;
            fscanf(fp,"%d %d %f %d", &vessel,&stop, &speed, &hold);
            if ( j == 0 )
                {
                start = stop;
                hh = xx / 60;
                mm = xx % 60;
                l++;
                /*
                SET_navy;
                if (mm < 10)
                    printf("  %2d:0%1d", hh, mm);
                else
                    printf("  %2d:%2d", hh, mm);
                */
                goto next_j;
                }
            for ( k = 0; k < num_tt; k++)
                if (vessel == ves[k] && stop == to[k] && pre == from[k])
                    {
                    m = k;
                    goto find_out;
                    }
            l++;
            /*
            printf("[%2d %2d]", pre, stop);
            */
            goto next_j;
find_out:
            if (xx >= peak_tm[j] && xx <= off_peak_tm[j])
                default_hold = peak_load[j];
            else
                default_hold = off_peak_load[j];
            xx = xx + hold+default_hold+travel_tm[m]*speed;
            if (start == stop)
                {
                dept_tm[i][n13[i]] = xx;
                n13[i]++;
                }
            hh = xx / 60;
            mm = xx % 60;
            if ( hh >= 24 )
                {
                next_day = 1;
                hh -= 24;
                xx -= 24;
                }
            l++;
            /*
            if (next_day) COLOR_off;
            if (start == stop) SET_navy;
            if (next_day && start == stop) SET_blue;
            if (start == stop)
                {
            if (mm < 10)
                printf("  %2d:0%1d", hh, mm);
            else
                printf("  %2d:%2d", hh, mm);

            if ((l % 10) == 9) printf("\n");
                }
            */
next_j:
            pre = stop;
            }
        if ((num_stops % 10) != 0) printf("\n");
        }
    fclose(fp);
    return;
    }

void load_pktimes()
    {
    int i, j, l, m, n, hh, mm;
    static char stimes[12] = "pkhour";
    FILE *fp, *fopen();

    if ((fp = fopen(stimes,"r"))== NULL)
        {
        file_msg(stimes);   return;
        }
    for ( i = 0; i < NUM_TERMINAL; i++)
        {
        peak_tm[i] = 0;
        off_peak_tm[i] = 0;
        fscanf(fp,"%s",stimes);
        l = strlen(stimes);
        hh = 0;
        mm = 0;
        for ( m = 0; m < l; m++)
            {
            n = m;
            if (stimes[m] == 58)  break;
            hh=hh*10+stimes[m]-48;
            }
        for ( m = n+1; m < l; m++)
            mm=mm*10+stimes[m]-48;
        peak_tm[i] = 60 * hh + mm;
        fscanf(fp,"%s",stimes);
        if (stimes[0] == 80 || stimes[0] == 112)
            peak_tm[i] += 720;
        fscanf(fp,"%s",stimes);
        l = strlen(stimes);
        hh = 0;
        mm = 0;
        for ( m = 0; m < l; m++)
            {
            n = m;
            if (stimes[m] == 58)  break;
            hh=hh*10+stimes[m]-48;
            }
        for ( m = n+1; m < l; m++)
            mm=mm*10+stimes[m]-48;
        off_peak_tm[i] = 60 * hh + mm;
        fscanf(fp,"%s",stimes);
        if (stimes[0] == 80 || stimes[0] == 112)
            off_peak_tm[i] += 720;
        }
    fclose(fp);
    return;
    }

void load_ltimes()
    {
    int i, j, k;
    static char stime[15] = "tload.dat";
    FILE *fp, *fopen();

    if ((fp = fopen(stime,"r"))== NULL)
        {
        file_msg(stime);   return;
        }
    for ( i = 0; i < NUM_TERMINAL; i++)
        fscanf(fp,"%d",peak_load[i]);
    for ( i = 0; i < NUM_TERMINAL; i++)
        fscanf(fp,"%d",off_peak_load[i]);
    fclose(fp);
    return;
    }
```

```
/* user */
#define cleanscreen printf("\033[2J");
#define poscur(r,c) printf("\033[%d;%dH",r,c);
#define SET_black   printf("\033[1;30m");
#define SET_red     printf("\033[1;31m");
#define SET_green   printf("\033[1;32m");
#define SET_yellow  printf("\033[1;33m");
#define SET_blue    printf("\033[1;34m");
#define SET_cyan    printf("\033[1;35m");
#define SET_navy    printf("\033[1;36m");
#define SET_white   printf("\033[1;37m");
#define COLOR_off   printf("\033[m");
#define beep_x      printf("\07");
#define set_txt_blnk(f)   printf("\033[5;%dm",f);
/*  front text/background color */
#define set_screen(f,b)   printf("\033[1;%d;%dm",f,b);
/* beep */
beep(int num)
    {
    while(num--)
        beep_x;
    }
/* user1 */
slbox(r1, c1, words, upper, lower)
    int r1, c1, words, lower, upper;
    {
    int j, i;
    poscur(r1,c1);          /* first line */
    printf("%c",218);
    for(j = 0; j < words; printf("%c",196), j++);
    printf("%c\n",191);     r1++;
    if ( upper != 0 )       /* heading line */
        {
        for (i = 0; i < upper; i++)
            {
            poscur(r1,c1);      printf("%c",179);
            poscur(r1,c1+words+1);  printf("%c\n",179);         r1++;
            }
        if ( lower != 0 )
            {
            poscur(r1,c1);            printf("%c",198);
            for(j = 0; j < words; printf("%c",205), j++);
            printf("%c\n",181);     r1++;
            }
        }
    if ( lower != 0 )       /* body line */
        {
        for (i = 0; i < lower; i++)
            {
            poscur(r1,c1);      printf("%c",179);
            poscur(r1,c1+words+1);  printf("%c\n",179);     r1++;
            }
        }
    poscur(r1,c1);          /* last line */
    printf("%c",192);
    for(j = 0; j < words; printf("%c",196), j++);       printf("%c",217);
    }
dlbox(r1, c1, words, upper, lower)
    int r1, c1, words, lower, upper;
    {
    int j, i;
    poscur(r1,c1);  printf("%c",201);
    for(j = 0; j < words; printf("%c",205), j++);
    printf("%c\n",187);     r1++;
    if ( upper != 0 )   {
        for (i = 0; i < upper; i++)     {
            poscur(r1,c1);            printf("%c",186);
            poscur(r1,c1+words+1);  printf("%c\n",186);     r1++;
            }
        if ( lower != 0 )   {
            poscur(r1,c1);            printf("%c",204);
            for(j = 0; j < words; printf("%c",205), j++);
            printf("%c\n",185);     r1++;
            }
        }
    if ( lower != 0 )
        for (i = 0; i < lower; i++)     {
            poscur(r1,c1);      printf("%c",186);
            poscur(r1,c1+words+1);  printf("%c\n",186);     r1++;
            }
    poscur(r1,c1);      printf("%c",200);
    for(j = 0; j < words; printf("%c",205), j++);   printf("%c",188);
    }
/* user2 */
file_msg(name)
    char name[15];
    {
    printf("\n\n\tfile %s is not found, please check...\n\n",name);
    beep(1);        getchar();
    }
c_error(int r, char c)
    {
    set_screen(37,41);      poscur(r,10);       beep(2);
    printf("invalid input '%c', please try again...\n",c);  COLOR_off;
    }
error(int r, int d)
    {
    set_screen(31,47);      poscur(r,10);       beep(2);
    printf("input '%d' is an error, please try again!!",d); COLOR_off;
    }
enter(int r)
    {
    set_screen(37,42);      poscur(r,50);
    printf("Press <Enter> to continue");    COLOR_off;      getchar();
    }
choose(r, c)
    int r, c;
{
    SET_green;
    poscur(r,c);
    printf("Your choice is >>     ");
    COLOR_off;
}
subscrn(fname)
    char fname[20];
    {
    char line[80];
    int i;
    FILE *fp, *fopen();
    if ((fp = fopen(fname,"r")) == NULL)
        {
        file_msg(fname);    return(0);
        }
    fgets(line, 79, fp);
    while(feof(fp)==0)
        {
        printf("%s", line);     fgets(line, 80, fp);
        }
    fclose(fp);
    }
submenu(fname,f,b)
    char fname[20];
    int f, b;
    {
    char line[80];
    int upper, lower, k, r, c, len, i;
    FILE *fp, *fopen();

    if ((fp = fopen(fname,"r")) == NULL)
        {
        printf("file %s is not found, please check....\n",fname);
        getchar();
        }
    fscanf(fp, "%d %d %d %d", &r, &c, &upper, &lower);
    fgets(line, 60, fp);
    if ( upper != 0 )
        {
        for ( k = 0; k < upper; k++)
            {
            fgets(line,60,fp);
            len = strlen(line);
            for ( i = len-1; i < 45; line[i] = 32, i++);
            set_screen(f,b);
            poscur(r,c);
            printf("%-45.45s\n", line);
            COLOR_off;
            r++;
            }
        r+=1;
        }
    if ( lower != 0 )
        {
        for ( k = 0; k < lower; k++)
            {
            fgets(line,60,fp);
            len = strlen(line);
            for ( i = len-1; i < 45; line[i] = 32, i++);
            set_screen(f,b);
            poscur(r,c);
            printf("%-45.45s\n", line);
            COLOR_off;
            r++;
            }
        }
    fclose(fp);
    }
/* user3 */
void sleep()  /* ZENITH 386/20 */
    {
    long i, j, d;  /* 80917 */
    for ( j = 1; j <= 10; j++)
        for ( i = 1; i <= 82510; i++)
            d = i % j;  /* 82509.432038 */
    return;
    }
int today_is(int yy, int mm, int dd)
    {
    int i = 0, leap = 0, day = 0;
    static int mon[13] = { 0, 31,28,31,30,31,30,31,31,30,31,30,31};
    long j= 0;
    for ( i = 1; i < yy; i++)
        leap += leap_year(i);
    j = 365 * (yy-1) + leap;
    day = j % 7;
    mon[2] = 28+leap_year(yy);
    for ( i = 0; i < mm; i++)
        dd += mon[i];
    day += dd;
    day = (day+1) % 7;
    return(day);
    }
int leap_year(int yr)
    {
    int feb;
    if ((yr/400)==(yr/400.))
        feb = 1;
    else
        if ((yr/4)==(yr/4.))
            if ((yr/100)==(yr/100.))
                feb = 0;
            else
                feb = 1;
        else
            feb = 0;
    return(feb);
    }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "crew_aux.c"

#define NUM_SCENARIO 10
#define NUM_VESSEL    3
#define NUM_TERMINAL 25
#define NUM_ROUTE    15
#define NUM_SHIFT     8

int num_scenario;
int current_scenario;
char fname[NUM_SCENARIO][12];
int current_route;
int current_vessel;
int current_shift;
char scenario_name[NUM_SCENARIO][24];

struct ScenStruct
    {
    char name[24];
    char description[36];
    int  num_route;
    struct RouteStruct
        {
        int  num;
        int  num_vessel;
        struct vesstruct
            {
            int num;
            int  num_shift;
            struct ShiftStruct
                {
                char name[24];
                int type;
                int dayon;
                int dayoff;
                float maxlen;
                float minlen;
                float days;
                float dailyover;
                float daysover;
                float spread;
                float minoff;
                int dailypen;
                int dayspen;
                int spreadpen;
                int report;
                int tieup;
                } Shift[NUM_SHIFT];
            } Vessel[NUM_VESSEL];
        } Route[NUM_ROUTE];
    } Scenario;

int num_route;
char route_name[NUM_ROUTE][32];
int num_vessel;
char vessel_name[NUM_VESSEL][12];
int num_terminal;
char terminal_name[NUM_VESSEL][12];
char general_entry[40];

void load_terminal(void);
void load_route(void);
void load_vessel(void);
void read_scenario(int,int);
void load_scenario(char *);
void list_scenario(void);
void save_scenario(void);
void delete_scenario(void);
void watch1(int,int);
void watch2(int *,int *);
void watch3(int,int);
void watch4(int,int);
void watch5(int,int);
void watch7(int,int);
void watch8(int,int);

main()
```

```
{
    int f = 33, b = 41;
    char answer;

    current_scenario = 0;
    current_route    = 0;
    current_vessel   = 0;
    current_shift    = 0;
    num_scenario     = 0;
    load_terminal();
    enter(25);
    load_route();
    enter(25);
    load_vessel();
    enter(25);
    read_scenario(f,b);
begin:
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 9);
    submenu("watch0.mnu",f,b);
input:
    choose(16,24);
    poscur(16,44);
    answer=getch();
    switch(answer-48)
        {
        case 1 : watch1(f,b);           break;
        case 2 : watch2(&f,&b);         break;
        case 3 : watch3(f,b);           break;
        case 4 : watch4(f,b);           break;
        case 5 : watch5(f,b);           break;
        case 6 : read_scenario(f,b);    break;
        case 7 : watch7(f,b);           break;
        case 8 : watch8(f,b);           break;
        case 9 : COLOR_off;
                 printf("\n\n");
                 exit(0);               break;
        default: c_error(20, answer);   goto input;
        }
    goto begin;
}

void watch11(void);
void watch12(void);
void watch13(void);
void watch14(void);
void watch15(void);
void watch16(void);
void watch17(void);

#include "watch1.c"

void watch1(f,b)
    int f, b;
    {
    char answer;

begin:
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 8);
    submenu("watch1.mnu",f,b);
input:
    choose(15,24);
    poscur(15,44);
    answer=getch();
    switch(answer-48)
        {
        case 1 : watch11();   break;
        case 2 : watch12();   break;
        case 3 : watch13();   break;
        case 4 : watch14();   break;
        case 5 : watch15();   break;
        case 6 : watch16();   break;
        case 7 : watch17();   break;
        case 9 : return;      break;
        default: c_error(18, answer);    goto input;
        }
    goto begin;
}

void watch2(f,b)
    int *f, *b;
    {
    COLOR_off;
    cleanscreen;
    set_screen(37,40);
    subscrn("watch2.mnu",f,b);
input_f:
    poscur(14,31);
    scanf("%d",&*f);
    getchar();
    if (*f == -1)
        {
        *f = 33;
        *b = 41;
        return;
        }
    if (*f > 0 && *f <= 8 )
        *f = *f+29;
    else
        {
        error(16,*f);
        goto input_f;
        }
input_b:
    poscur(14,56);
    scanf("%d",&*b);
    getchar();
    if (*b > 0 && *b <= 8 )
        *b = *b+39;
    else
        {
        error(16,*b);
        goto input_b;
        }
    set_screen(*f,*b);
}

void watch31(void);
void watch32(void);
void watch33(void);
void watch34(void);
void watch35(void);
void watch36(void);

#include "watch3.c"

void watch3(f,b)
    int f, b;
    {
    char answer;

begin:
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 7);
    submenu("watch3.mnu",f,b);
input:
    choose(13,24);
    poscur(13,44);
    answer=getch();
    switch(answer-48)
        {
        case 1 : watch31();   break;
        case 2 : watch32();   break;
        case 3 : watch33();   break;
        case 4 : watch34();   break;
        case 5 : watch35();   break;
        case 6 : watch36();   break;
        case 9 : return;      break;
        default: c_error(16,answer);    goto input;
        }
    goto begin;
}
```

```
#include "watch4.c"

void watch4(f,b)
    int f, b;
    {
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 0);
    submenu("watch4.mnu",f,b);
    show_schedule();
    return;
    }

void watch5(f,b)
    int f, b;
    {
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 9);
    submenu("watch5.mnu",f,b);
    produce_report();
    return;
    }

void watch7(f,b)
    int f, b;
    {
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 0);
    submenu("watch7.mnu",f,b);
    save_scenario();
    enter(10);
    return;
    }

void watch8(f,b)
    int f, b;
    {
    COLOR_off;
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, num_scenario+1);
    submenu("watch8.mnu",f,b);
    set_screen(f,b);
    delete_scenario();
    return;
    }
```

```
void watch11()      /* choose a route */
    {
    int i, item;

    COLOR_off;      cleanscreen;
    SET_navy;       poscur(5,10);
    printf("Choose Route to Schedule Sub Menu");
    SET_red;
    poscur(7,15);     printf(" 0) New Scrnario Route          ");
    for ( i = 0; i < Scenario.num_route; i++)
        {
        poscur(8+i,15);     printf("%2d) %-27.27s", i+1,Scenario.Route[i].name);
        }
input:
    choose(9+Scenario.num_route,10);    poscur(9+Scenario.num_route,30);
    scanf("%d", &item);
    getchar();
    if ( item == 0) goto new;
    if (item < 0 || item > Scenario.num_route)
        {
        error(19, item);   goto input;
        }
    current_route = item-1;      return;
new:
    current_route = Scenario.num_route;
    Scenario.num_route++;
    SET_green;
    printf(" New Route Name ==> ");     COLOR_off;
    gets(Scenario.Route[current_route].name);
    Scenario.Route[current_route].num_shift = 0;
    return;
    }

char route_name[14][30];
char route_term[14][10];
char *name[] = {    "                              " ,
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth",
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth",
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth",
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth",
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth",
    "Vashon Special Schedule 2",    "Fauntleroy-Vashon-Southworth",
    "Fauntleroy-Vashon-Southworth", "Fauntleroy-Vashon-Southworth"  };
char *term[] = { "        ", "Fall  89", "Fall  89", "Fall  89", "Fall  89",
     "Fall  89", "Fall  89", "Fall  89", "Fall  89", "Fall  89",
     "Fall  89", "        ", "Fall  89", "Fall  89", "Fall  89" };

void watch12_base(void);
void watch12_day(void);
void watch12_clear(void);

void watch12()      /* assign vessel */
    {
    char answer;
    int i;

    watch12_base();
begin:
    COLOR_off;
    cleanscreen;
    SET_yellow;
    printf("\n");
    subscrn("watch12.mnu");
    SET_navy;
    poscur(2,10);
    printf("Assign Vessel Schedule to Route Sub Menu");
    COLOR_off;
    /* display vessel schedule */
    SET_white;
    for ( i = 0; i < 7; i++)
        {
        poscur(4+i,12); printf("%-30.30s", route_name[i]);
        poscur(4+i,49); printf("%-10.10s", route_term[i]);
        }
    for ( i = 7; i < 14; i++)
        {
        poscur(5+i,12); printf("%-30.30s", route_name[i]);
        poscur(5+i,49); printf("%-10.10s", route_term[i]);
        }
    SET_red;
    dlbox(1,8,45,1,0);
    dlbox(19,4,55,2,0);
    SET_navy;
    poscur(20,5);
    printf("  (1) Choose Base Schedule    (2) Change Day Schedule  ");
    poscur(21,5);
    printf("  (3) Clear Schedule          (4) Previous Menu        ");
input:
    SET_green;      choose(23,10);        poscur(23,30);
    COLOR_off;      answer=getch();       putch(answer);
    switch(answer-48)
        {
        case 1 : watch12_base();    break;
        case 2 : watch12_day();     break;
        case 3 : watch12_clear();   break;
        case 4 : return;
        default: c_error(24,answer);    COLOR_off;      goto input;
        }
    goto begin;
    }
void watch12_base()
    {
    int i;
    for ( i = 0; i < 14; i++)
        {
        strcpy(route_name[i],name[i+1]);
        strcpy(route_term[i],term[i+1]);
        }
    return;
    }
void watch12_day()
    {
    return;
    }
void watch12_clear()
    {
    int i;
    for ( i = 0; i < 14; i++)
        {
        strcpy(route_name[i],name[0]);
        strcpy(route_term[i],term[0]);
        }
    return;
    }
void watch13()      /* choose crew shift */
    {
    int i, item;
    int now = current_route;

    COLOR_off;      cleanscreen;
    SET_navy;       poscur(5,10);
    printf("Choose Watch Group Sub Menu");
    SET_red;
    poscur(7,15);     printf(" 0) New Watch Crew           ");
    for ( i = 0; i < Scenario.Route[current_route].num_shift; i++)
        {
        poscur(8+i,15);
        printf("%2d) %-24.24s",i+1,Scenario.Route[current_route].Shift[i].name);
        }
input:
    choose(9+Scenario.Route[current_route].num_shift,10);
    poscur(9+Scenario.Route[current_route].num_shift,30);
    scanf("%d", &item);
    getchar();
    if ( item == 0) goto new;
    if (item < 0 || item > Scenario.Route[current_route].num_shift)
        {
        error(19, item);   goto input;
        }
    current_shift = item-1;     return;
new:
    current_shift = Scenario.Route[current_route].num_shift;
    Scenario.Route[current_route].num_shift++;
    SET_green;
    printf(" New Shift Name ==> ");     COLOR_off;
    gets(Scenario.Route[current_route].Shift[current_shift].name);
    return;
    }
void watch14()      /* route level scheduling */
    {
```

```
    COLOR_off;
    cleanscreen;
    SET_yellow;
    subscrn("watch14.mnu");
    SET_green;
    poscur(4,8);        putch(88);
    poscur(5,8);        putch(88);
    poscur(6,8);        putch(32);
    poscur(7,8);        putch(32);
    poscur(8,8);        putch(32);
    poscur(9,8);        putch(32);
    poscur(11,6);       putch(32);
    poscur(12,6);       putch(32);
    poscur(13,6);       putch(32);
    poscur(17,28);      putch(82);
    poscur(17,44);      putch(82);
    poscur(17,58);      putch(32);
    poscur(18,28);      putch(84);
    poscur(18,44);      putch(84);
    poscur(18,58);      putch(32);
    poscur(19,28);      putch(32);
    poscur(19,44);      putch(32);
    poscur(19,58);      putch(66);
    enter(24);
    return;
    }

void watch15_basic(int,int);
void watch15_OK(void);
void watch15_ED(void);

void watch15()      /* define a shift */
    {
    char answer;
    int ii = 1, jj = 0;

begin:
    watch15_basic(ii,jj);
    SET_navy;
    poscur(24,1);
    printf(" (1) Accept Proposed Shift  (2) Edit Shifts ");
    if (ii == 1)
        printf(" (3) Next Week\n");
    else
        printf(" (3) Previous Week\n");
    printf(" (4) Next Vessel   (5) Previous Menu");
input:
    SET_green;       choose(25,40);         poscur(25,60);
    COLOR_off;       answer=getch();        putch(answer);
    switch(answer-48)
        {
        case 1 : watch15_OK();      break;
        case 2 : watch15_ED();      break;
        case 3 : ii = ii % 2 + 1;   break;
        case 4 : jj = (jj+1) % 22;   ii = 1;     break;
        case 5 : return;
        default: c_error(24,answer);    COLOR_off;      goto input;
        }
    goto begin;
    }

void watch15_basic(int ii, int jj)
    {
    int i, j;
    static char time[13][5] = {" 7:30", " 8:50", "10:10", "11:30", "12:50",
    "14:10", "15:30", "16:50", "18:10", "19:30", "20:50", "22:10", "23:30"};
      static char vessel[22][12] = { "Cathlamet", "Chelan", "Elwha", "Evergreen",
     "Hiyu", "Hyak", "Illahee", "Issaquah", "Kaleetan", "Kitsap", "Kittitas",
     "Klahowya", "Klickitat", "Nisqually", "Olympic", "Quinault",
     "Rhododendron", "Sealth", "Spokane", "Tillikum", "Walla_Walla", "Yakima"};

    COLOR_off;
    cleanscreen;
    SET_yellow;
    subscrn("watch15.mnu");
    COLOR_off;
    set_screen(37,44);
      poscur(17,15);       printf(" %-12.12s ",vessel[jj]);
    set_screen(37,42);
    poscur(17,48);      printf(" Week %d ",ii);
    COLOR_off;
    SET_white;
    for ( i = 0; i < 7; i++)
        {
        poscur(2+i*3,64); printf(" 7:30-15:30");
        poscur(3+i*3,64);
        if ( i < 5 )
            printf("15:30-23:30");
        else
            printf("15:30-24:50");
        }
    for ( i = 0; i < 13; i++)
        for ( j = 0; j < 7; j++)
            {
            poscur(2+i,4+8*j); printf("%-5.5s", time[i]);
            }
    SET_cyan;
    for ( j = 0; j < 7; j++)
        {
        poscur(20,4+8*j); printf("%-5.5s", " 8:00");
        poscur(21,4+8*j); printf("%-5.5s", " 8:00");
        }
    return;
    }
void watch15_OK()
    {
    cleanscreen;
    poscur(12,30);
    set_txt_blnk(35);   printf("Accept Proposed Shifts");
    COLOR_off;
    enter(20);
    poscur(20,40);
    printf("                              ");
    poscur(15,30);
    printf("please wait....");
    delay(2500);
    return;
    }
void watch15_ED()
    {
    return;
    }
void watch15_vessel()
    {
    return;
    }

char shift_name[14][8];
char *xname[] = {  "        ",  "CBFEJH  ",  "ABFEJHC ",  "ABDEJHF ",
                   "ABDEGHJ ",  "ACDEGHB ",  "ACDFGHE ",  "ACDFGJH ",
                   "ACDFGJ  ",  "ABDFGJC ",  "ABDEGJF ",  "ABDEGHJ ",
                   "CBDEGHA ",  "CBFEGHD ",  "CBFEJHG "  };

void watch16_basic(void);
void watch16_begin(void);
void watch16_delete(void);
void watch16_move(void);
void watch16_list(void);
void watch16_ext(void);

void watch16()      /* assign crew */
    {
    char answer;
    int i, j;

    watch16_basic();
begin:
    COLOR_off;
    cleanscreen;
    SET_navy;
    subscrn("watch16.mnu");
    SET_white;
    for ( i = 0; i < 7; i++)
        {
        for ( j = 0; j < 8; j+=2)
            {
            poscur(4+i,9+(j/2)*20);     putchar(shift_name[i][j]);
            poscur(4+i,13+(j/2)*20);    putchar(shift_name[i][j+1]);
            }
        }
    for ( i = 7; i < 14; i++)
        {
        for ( j = 0; j < 8; j+=2)
            {
            poscur(5+i,9+(j/2)*20);     putchar(shift_name[i][j]);
            poscur(5+i,13+(j/2)*20);    putchar(shift_name[i][j+1]);
            }
        }
    poscur(20,48);  printf("%-28.28s", Scenario.Route[current_route].name);
    SET_yellow;
    poscur(22,5);
    printf("(1) Begin Watch   (2) Delete Watch    (3) Move Watch   ");
    poscur(23,5);
    printf("(4) Crew Summary  (5) External Crews  (6) Previous Menu");
input:
    SET_green;      choose(24,10);          poscur(24,30);
    COLOR_off;      answer=getch();         putch(answer);
    switch(answer-48)
        {
        case 1 : watch16_begin();   break;
        case 2 : watch16_delete();  break;
        case 3 : watch16_move();    break;
        case 4 : watch16_list();    break;
        case 5 : watch16_ext();     break;
        case 6 : return;
        default: c_error(25,answer);    COLOR_off;      goto input;
        }
    goto begin;
    }

void watch16_basic()
    {
    int i;

    for ( i = 0; i < 14; i++)
        {
        strcpy(shift_name[i],xname[i+1]);
        }
    return;
    }
void watch16_begin()
    {
    return;
    }
void watch16_delete()
    {
    return;
    }
void watch16_move()
    {
    return;
    }
void watch16_list()
    {
    static char tt[7][3] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
    int i, j, r, c;

    COLOR_off;
    cleanscreen;
    SET_navy;
    subscrn("watch16a.mnu");
    SET_yellow;    poscur(1,25);   printf("CREW SUMMARY LIST");
    for ( i = 0; i < 9; i++)
        {
        poscur(5+i,7);  printf("%c",65+i);
        poscur(15+i,7); printf("%c",65+i);
        }
    for ( i = 0; i < 7; i++)
        {
        SET_green;
        poscur(3,12+8*i);   printf("%3.3s", tt[i]);
        SET_yellow;
        poscur(4,11+8*i);   printf("1");
        poscur(4,13+8*i);   printf("2");
        poscur(4,15+8*i);   printf("3");
        }
    SET_white;
    poscur(2,2);    putch(218);
    for ( i = 0; i < 62; putch(196), i++);   putch(191);
    for ( i = 0; i < 7; i++)
```

```
        {
        poscur(2,9+8*i);
        putch(194);
        }
    for ( i = 0; i < 11; i++)
        {
        poscur(3+i,2);  putch(179);
        for ( j = 0; j < 8; putch(179), j++)
            poscur(3+i,9+8*j);
        }
    poscur(14,2);   putch(198);
    for ( i = 0; i < 62; putch(205), i++);   putch(181);
    for ( i = 0; i < 7; putch(216), i++)
        poscur(14,9+8*i);
    for ( i = 11; i < 20; i++)
        {
        poscur(4+i,2);  putch(179);
        for ( j = 0; j < 8; putch(179), j++)
            poscur(4+i,9+8*j);
        }
    poscur(24,2);    putch(192);
    for ( i = 0; i < 62; putch(196), i++);   putch(217);
    for ( i = 0; i < 7; putch(193), i++)
        poscur(24,9+8*i);
    enter(25);
    return;
    }
void watch16_ext()
    {
    return;
    }

void watch17()      /* show schedule */
    {
    printf("\n\n\tShow Scheduling Status \n");
    getchar();
    return;
    }
```

```
void watch31()
    {
    char answer;

    COLOR_off;
    cleanscreen;
    SET_navy;
    poscur(5,10);
    printf("Scenario Name/Description Sub_Menu");
    SET_red;
    poscur(7,5);
    printf("Scenario Name ==> ");
    poscur(8,5);
    printf("Scenario Description ==> ");
    SET_green;
    poscur(7,23);
    printf("%s",Scenario.name);
    poscur(8,30);
    printf("%s",Scenario.description);
input:
    set_screen(37,42);
    poscur(10,20);
    printf("  Do you want to change them ? (y/n)    ");
    SET_yellow;
    poscur(10,58);
    scanf("%c", &answer);
    getchar();
    if (answer == 78 || answer == 110) return;
    if (answer == 89 || answer == 121) goto change;
    c_error(12,answer);
    goto input;
change:
    COLOR_off;
    poscur(7,23);
    gets(Scenario.name);
    poscur(8,30);
    gets(Scenario.description);
    return;
    }
void watch32()
    {
    int   r, c, i, j, num_route, jj;
    char cc;
    int now = current_route;

    r = 12;
    num_route = Scenario.num_route;
    if (num_route != 0)
        {
        set_screen(33,42);
        for ( i = 0; i < num_route; i++)
            {
            poscur(r+i,25);
            printf("  %2d) %-30.30s\n",i+1, Scenario.Route[i].name);
            }
        COLOR_off;
        }
    set_screen(33,42);
    dlbox(11, 24, 35, 0, num_route);
    enter(22);
    return;
    }

void watch33()
    {
    int   r, c, i, j, num_type, jj;
    char cc;
    int now = current_route;

    r = 13;
    num_type = Scenario.Route[now].num_shift;
    if (num_type != 0)
        {
        set_screen(33,42);
        for ( i = 0; i < num_type; i++)
            {
            poscur(r+i,25);
            printf("  %2d) %-30.30s\n",i+1, Scenario.Route[now].Shift[i].name);
            }
        COLOR_off;
```

```
            }
        set_screen(33,42);
        dlbox(11, 24, 35, 0, num_type+1);
        poscur(12,25);
        printf("   0) New Shift Type               ");
        COLOR_off;
        choose(15+num_type, 30);
        poscur(15+num_type, 50);
        scanf("%d",&c);
        getchar();
        if ( c < 0 || c > num_type ) return;
        cleanscreen;
        SET_yellow;
        subscrn("watch33.mnu");
        COLOR_off;
        if ( c == 0)
            {
            poscur(2,30);
            gets(general_entry);
            strcpy(Scenario.Route[now].Shift[num_type].name,general_entry);
type_n1:
            Scenario.Route[now].Shift[num_type].type = 1;
            poscur(4,13);   putchar(88);
            poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(32);    poscur(4,13);
            cc = getch();   if (cc == 0)  cc = getch();
            if (cc == '\t' || cc == 77 || cc == 80)  goto type_n2;
            if (cc == 88 || cc == 120)   goto type_nx;
type_n2:
            Scenario.Route[now].Shift[num_type].type = 2;
            poscur(4,13);   putchar(32);
            poscur(5,13);   putchar(88);
            poscur(6,13);   putchar(32);    poscur(5,13);
            cc = getch();   if (cc == 0)  cc = getch();
            if (cc == '\t' || cc == 77 || cc == 80)  goto type_n3;
            if (cc == '\b' || cc == 72 || cc == 75)  goto type_n1;
            if (cc == 88 || cc == 120)   goto type_nx;
type_n3:
            Scenario.Route[now].Shift[num_type].type = 3;
            poscur(4,13);   putchar(32);
            poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(88);    poscur(6,13);
            cc = getch();   if (cc == 0)  cc = getch();
            if (cc == '\b' || cc == 72 || cc == 75)  goto type_n2;
            if (cc == '\t' || cc == 77 || cc == 80)  goto type_nx;
            if (cc == 88 || cc == 120)   goto type_nx;
type_nx:
            poscur(8,38);   scanf("%d",&Scenario.Route[now].Shift[num_type].dayon);
            poscur(9,38);   scanf("%d",&Scenario.Route[now].Shift[num_type].dayoff);
            poscur(10,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].maxlen);
            poscur(11,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].minlen);
            poscur(12,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].days);
            poscur(13,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].dailyover);
            poscur(14,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].daysover);
            poscur(15,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].minoff);
            poscur(16,38);  scanf("%f",&Scenario.Route[now].Shift[num_type].spread);
            poscur(17,38);  scanf("%d",&Scenario.Route[now].Shift[num_type].dailypen);
            poscur(18,38);  scanf("%d",&Scenario.Route[now].Shift[num_type].dayspen);
            poscur(19,38);  scanf("%d",&Scenario.Route[now].Shift[num_type].spreadpen);
            poscur(20,38);  scanf("%d",&Scenario.Route[now].Shift[num_type].report);
            poscur(21,38);  scanf("%d",&Scenario.Route[now].Shift[num_type].tieup);
            getchar();      num_type++;
            }
        else
            {
            SET_green;
            poscur(2,30);   printf("%-30.30s", Scenario.Route[now].Shift[c-1].name);
            j = Scenario.Route[now].Shift[c-1].type;
            poscur(3+j,13); putch(88);
            poscur(8,38);   printf("%-4d", Scenario.Route[now].Shift[c-1].dayon);
            poscur(9,38);   printf("%-4d", Scenario.Route[now].Shift[c-1].dayoff);
            poscur(10,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].maxlen);
            poscur(11,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].minlen);
            poscur(12,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].days);
            poscur(13,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].dailyover);
            poscur(14,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].daysover);
            poscur(15,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].minoff);
            poscur(16,38);  printf("%4.1f", Scenario.Route[now].Shift[c-1].spread);
            poscur(17,38);  printf("%-4d", Scenario.Route[now].Shift[c-1].dailypen);
            poscur(18,38);  printf("%-4d", Scenario.Route[now].Shift[c-1].dayspen);
        poscur(19,38);  printf("%-4d", Scenario.Route[now].Shift[c-1].spreadpen);
        poscur(20,38);  printf("%-4d", Scenario.Route[now].Shift[c-1].report);
        poscur(21,38);  printf("%-4d", Scenario.Route[now].Shift[c-1].tieup);
        COLOR_off;
        poscur(2,30);   gets(general_entry);
        strcpy( Scenario.Route[now].Shift[c-1].name,general_entry);
        poscur(2,30);   printf("%-30.30s", Scenario.Route[now].Shift[c-1].name);
        jj = 0;
type_s1:
        if (jj != 0)
            {
            poscur(4,13);   putchar(88);    poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(32);
            }
        poscur(4,13);       cc = getch();   if (cc == 0)  cc = getch();
        if (cc == '\t' || cc == 77 || cc == 80)  goto type_s2;
        jj++;
        if (cc == 88 || cc == 120)
            {
            poscur(4,13);   putchar(88);    poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(32);    Scenario.Route[now].Shift[c-1].type = 1;
            goto type_sx;
            }
type_s2:
        if (jj != 0)
            {
            poscur(4,13);   putchar(32);    poscur(5,13);   putchar(88);
            poscur(6,13);   putchar(32);
            }
        poscur(5,13);       cc = getch();   if (cc == 0)  cc = getch();
        if (cc == '\t' || cc == 77 || cc == 80)  goto type_s3;
        if (cc == '\b' || cc == 72 || cc == 75)  goto type_s1;
        jj++;
        if (cc == 88 || cc == 120)
            {
            poscur(4,13);   putchar(32);    poscur(5,13);   putchar(88);
            poscur(6,13);   putchar(32);    Scenario.Route[now].Shift[c-1].type = 2;
            goto type_sx;
            }
type_s3:
        if (jj != 0)
            {
            poscur(4,13);   putchar(32);    poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(88);
            }
        poscur(6,13);       cc = getch();   if (cc == 0)  cc = getch();
        if (cc == '\b' || cc == 72 || cc == 75)  goto type_s2;
        if (cc == '\t' || cc == 77 || cc == 80)  goto type_sx;
        jj++;
        if (cc == 88 || cc == 120)
            {
            poscur(4,13);   putchar(32);    poscur(5,13);   putchar(32);
            poscur(6,13);   putchar(88);    Scenario.Route[now].Shift[c-1].type = 3;
            goto type_sx;
            }
type_sx:
        poscur(8,38);       scanf("%d",&Scenario.Route[now].Shift[c-1].dayon);
        poscur(8,38);       printf("%-4d",Scenario.Route[now].Shift[c-1].dayon);
        poscur(9,38);       scanf("%d",&Scenario.Route[now].Shift[c-1].dayoff);
        poscur(9,38);       printf("%-4d",Scenario.Route[now].Shift[c-1].dayoff);
        poscur(10,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].maxlen);
        poscur(10,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].maxlen);
        poscur(11,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].minlen);
        poscur(11,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].minlen);
        poscur(12,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].days);
        poscur(12,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].days);
        poscur(13,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].dailyover);
        poscur(13,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].dailyover);
        poscur(14,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].daysover);
        poscur(14,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].daysover);
        poscur(15,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].minoff);
        poscur(15,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].minoff);
        poscur(16,38);      scanf("%f",&Scenario.Route[now].Shift[c-1].spread);
        poscur(16,38);      printf("%4.1f",Scenario.Route[now].Shift[c-1].spread);
        poscur(17,38);      scanf("%d",&Scenario.Route[now].Shift[c-1].dailypen);
        poscur(17,38);      printf("%-4d",Scenario.Route[now].Shift[c-1].dailypen);
        poscur(18,38);      scanf("%d",&Scenario.Route[now].Shift[c-1].dayspen);
        poscur(18,38);      printf("%-4d",Scenario.Route[now].Shift[c-1].dayspen);
        poscur(19,38);      scanf("%d",&Scenario.Route[now].Shift[c-1].spreadpen);
        poscur(19,38);      printf("%-4d",Scenario.Route[now].Shift[c-1].spreadpen);
        poscur(20,38);          scanf("%d",&Scenario.Route[now].Shift[c-1].report);
        poscur(20,38);          printf("%-4d",Scenario.Route[now].Shift[c-1].report);
        poscur(21,38);          scanf("%d",&Scenario.Route[now].Shift[c-1].tieup);
        poscur(21,38);          printf("%-4d",Scenario.Route[now].Shift[c-1].tieup);
        getchar();
        }
    return;
    }

void watch34()
    {
    int   r, c, i, j, num_type;
    int now = current_route;

    num_type = Scenario.Route[now].num_shift;
    dlbox(11, 24, 35, 0, num_type+1);
    set_screen(33,42);
    poscur(12,25);
    printf("   0) No Delete Shift Type         ");
    for ( i = 0; i < num_type; i++)
        {
        poscur(13+i,25);
        printf("  %2d) %-30.30s", i+1, Scenario.Route[now].Shift[i].name);
        }
    COLOR_off;
    choose(14+num_type, 30);
    poscur(14+num_type, 50);
    scanf("%d",&c);
    getchar();
    if ( c == 0 || c > num_type )    return;
    Scenario.Route[now].num_shift = num_type-1;
    if (c == num_type) return;
    for ( i = c; i < num_type; i++)
        Scenario.Route[now].Shift[i-1] = Scenario.Route[now].Shift[i];
    return;
    }

void watch35()
    {
    int i, j, k, man[8][4];
    float cost[8][4];
    FILE *fp, *fopen();

    if ((fp = fopen("crewcost.dat","r"))== NULL)
        {
        file_msg("crewcost.dat");   enter(24);       return;
        }
    fscanf(fp,"%d",&k);
    COLOR_off;
    cleanscreen;
    SET_navy;
    poscur(3,25);
    printf("Crew Complements/Cost");
    SET_yellow;
    dlbox(2,23,23,1,0);
    SET_cyan;
    poscur(6,1);
    subscrn("watch35.mnu");
    SET_white;
    for ( i = 0; i < k; i++)
        for ( j = 0; j < 4; j++)
            {
            fscanf(fp, "%d %f", &man[i][j], &cost[i][j]);
            poscur(10+i,22+j*13);
            printf("%2d  %5.2f", man[i][j], cost[i][j]);
            }
    fclose(fp);
    enter(24);
    }

void watch36()
    {
    COLOR_off;
    cleanscreen;
    subscrn("watch36.mnu");
    enter(25);
    return;
    }
```

```
void read_scenario(int f, int b)
    {
    char answer;
    FILE *fp, *fopen();
    int i, j, k, l;

    if ((fp = fopen("scenario.dat","r"))==NULL)
        {
        file_msg("scenario.dat");   enter(20);
        num_scenario = 0;
        return;
        }
    i = 1;
    fgets(general_entry,15,fp);
    while(feof(fp)==0)
        {
        l = strlen(general_entry) -1;
        strncpy(fname[i],general_entry,l);
        fgets(general_entry,35,fp);
        l = strlen(general_entry)-1;
        strncpy(scenario_name[i],general_entry,l);
        num_scenario = i;
        i++;
        fgets(general_entry,15,fp);
        }
    fclose(fp);
    cleanscreen;
    set_screen(f,b);
    dlbox(1, 14, 45, 1, 1+num_scenario);
    submenu("watchx.mnu",f,b);
    set_screen(f,b);
    for ( i = 1; i <= num_scenario; i++)
        {
        k = strlen(scenario_name[i]);
        poscur(4+i,15+k);
        for ( j = 15+k; j < 60; printf(" "), j++);
        poscur(4+i,15);
        printf("  %2d) %-25.25s", i, scenario_name[i]);
        }
    choose(13,24);
input_x:
    poscur(13,44);
    answer=getch();
    if (answer >= 48 && answer <= 48+num_scenario)
        {
        current_scenario = answer- 48;
        if (current_scenario == 0) return;
        load_scenario(fname[answer-48]);
        }
    else
        {
        c_error(14,answer);
        goto input_x;
        }
    return;
    }

void load_scenario(name)   /* Loadin a Scenario */
    char name[12];
    {
    int i, j, k, l;
    FILE *fp, *fopen();

    if ((fp = fopen(name,"r"))==NULL)
        {
        file_msg(name);     return;
        }
    fgets(general_entry,23,fp);
    l = strlen(general_entry)-1;
    strncpy(Scenario.name,general_entry,l);
    fgets(general_entry,59,fp);
    l = strlen(general_entry)-1;
    if ( l > 35 ) l = 35;
    strncpy(Scenario.description,general_entry,l);
    fscanf(fp,"%d", &Scenario.num_route);
    for ( i = 0; i < Scenario.num_route; i++)
        {
        fgetc(fp);
        fgets(general_entry,39,fp);
        l = strlen(general_entry)-1;
        if ( l > 31) l = 31;
        strncpy(Scenario.Route[i].name,general_entry,l);    fgetc(fp);
        fscanf(fp, "%d", &Scenario.Route[i].num_shift);
        for ( j = 0; j < Scenario.Route[i].num_shift; j++)
            {
            fgetc(fp);
            fgets(general_entry,39,fp);
            l = strlen(general_entry)-1;
            if ( l > 23 ) l = 23;
            strncpy(Scenario.Route[i].Shift[j].name,general_entry,l);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].type);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].dayon);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].dayoff);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].maxlen);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].minlen);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].days);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].dailyover);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].daysover);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].minoff);
            fscanf(fp,"%f", &Scenario.Route[i].Shift[j].spread);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].dailypen);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].dayspen);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].spreadpen);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].report);
            fscanf(fp,"%d", &Scenario.Route[i].Shift[j].tieup);
            }
        }
    return;
    }

void list_scenario()   /* List current Scenario */
    {
    int i, j;

    SET_yellow;
    printf("\n\n\tList Current Scenario\n\n");
    COLOR_off;
    printf("%s\n",Scenario.name);
    printf("%s\n",Scenario.description);
    printf("%5d\n", Scenario.num_route);
    for ( i = 0; i < Scenario.num_route; i++)
        {
        printf("%s\n",Scenario.Route[i].name);
        printf( "%5d\n", Scenario.Route[i].num_shift);
        for ( j = 0; j < Scenario.Route[i].num_shift; j++)
            {
            printf("%s\n",Scenario.Route[i].Shift[j].name);
            printf(" %4d", Scenario.Route[i].Shift[j].type);
            printf(" %4d", Scenario.Route[i].Shift[j].dayon);
            printf(" %4d", Scenario.Route[i].Shift[j].dayoff);
            printf(" %.2f", Scenario.Route[i].Shift[j].maxlen);
            printf(" %.2f", Scenario.Route[i].Shift[j].minlen);
            printf(" %.2f", Scenario.Route[i].Shift[j].days);
            printf(" %.2f", Scenario.Route[i].Shift[j].dailyover);
            printf(" %.2f", Scenario.Route[i].Shift[j].daysover);
            printf(" %.2f", Scenario.Route[i].Shift[j].minoff);
            printf(" %.2f", Scenario.Route[i].Shift[j].spread);
            printf(" %4d", Scenario.Route[i].Shift[j].dailypen);
            printf(" %4d", Scenario.Route[i].Shift[j].dayspen);
            printf(" %4d", Scenario.Route[i].Shift[j].spreadpen);
            printf(" %4d", Scenario.Route[i].Shift[j].report);
            printf(" %4d\n", Scenario.Route[i].Shift[j].tieup);
            }
        }
    return;
    }

void show_schedule()
    {
    printf("\n\n\tremark show_schedule ( unfinished option )\n");
    getchar();
    return;
    }

void produce_report()    /* ..... */
    {
    char answer;
again:
    choose(15,25);
    poscur(15,45);
    scanf("%c",&answer);
    getchar();
    switch(answer-48)
        {
        case  0: return;    break;
        case  1:
        case  2:
        case  3:
        case  4:
        case  5:
        case  6:
        case  7: /* printf("\n unfinished option\n"); */
                 break;
        default: c_error(22,answer);
                 goto again;
        }
    goto again;
    return;
    }

void save_scenario()
    {
    int i, j;
    char name[20], answer;
    FILE *fp, *fopen();

    if (current_scenario == 0)
        {
        printf("\t Current Scenario is a New Scneario\n");
input:
        printf("\tdata to be saved on file... ==> ");
        scanf("%s",name);
        getchar();
        if ((fp = fopen(name,"r")) != NULL)
            {
            printf("file %s is already existed, do you want to use other one ?", name);
            scanf("%c",&answer);    getchar();
            fclose(fp);
            if (answer == 89 || answer == 121) goto input;
            }
        fp = fopen("scenario.dat","a");
        current_scenario = num_scenario;
        fprintf(fp,"%-15.15s",name);
        strcpy(fname[num_scenario],name);
        strcpy(scenario_name[num_scenario],Scenario.name);
        fprintf(fp,"%-30.30s\n",Scenario.name);
        }
    fp = fopen(fname[current_scenario],"w");
    fprintf(fp,"%-24.24s\n",Scenario.name);
    fprintf(fp,"%-36.36s\n",Scenario.description);
    fprintf(fp,"%2d\n", Scenario.num_route);
    for ( i = 0; i < Scenario.num_route; i++)
        {
        fprintf(fp, "%-28.28s\n", Scenario.Route[i].name);
        fprintf(fp, "%5d\n", Scenario.Route[i].num_shift);
        for ( j = 0; j < Scenario.Route[i].num_shift; j++)
            {
            fprintf(fp,"%-25.25s\n",Scenario.Route[i].Shift[j].name);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].type);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].dayon);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].dayoff);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].maxlen);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].minlen);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].days);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].dailyover);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].daysover);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].spread);
            fprintf(fp," %.1f",  Scenario.Route[i].Shift[j].minoff);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].dailypen);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].dayspen);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].spreadpen);
            fprintf(fp," %d",  Scenario.Route[i].Shift[j].report);
            fprintf(fp," %d\n",  Scenario.Route[i].Shift[j].tieup);
            }
        }
     fclose(fp);
     printf("\n\n Current Scenario Schedule is saved on file %s", fname[current_scenario]);
     return;
     }
```

```
void delete_scenario()
    {
    char answer;
    FILE *fp, *fopen();
    int i, j, item, k, l;

    if ((fp = fopen("scenario.dat","r"))==NULL)
        {
        file_msg("scenario.dat");   enter(20);
        num_scenario = 0;
        return;
        }
    i = 0;
    fgets(general_entry,15,fp);
    while(feof(fp)==0)
        {
        i += 1;
        l = strlen(general_entry) -1;
        strncpy(fname[i],general_entry,l);
        fgets(general_entry,35,fp);
        l = strlen(general_entry)-1;
        strncpy(scenario_name[i],general_entry,l);
        num_scenario = i;
        fgets(general_entry,15,fp);
        }
    fclose(fp);
    for ( i = 1; i <= num_scenario; i++)
        {
        k = strlen(scenario_name[i]);
        poscur(4+i,15+k);
        for ( j = 15+k; j < 60; printf(" "), j++);
        poscur(4+i,15);
        printf("  %2d) %-25.25s", i, scenario_name[i]);
        }
    choose(13,24);
input_x:
    poscur(13,44);
    answer=getch();          putch(answer);
    if (answer >= 48 && answer <= 48+num_scenario)
        {
        item = answer- 48;
        if (item == 0) return;
        fp = fopen("scenario.dat","w");
        for ( i = 1; i <= num_scenario; i++)
            {
            if ( i != item )
                {
                fprintf(fp, "%-12.12s\n", fname[i]);
                fprintf(fp, "%-24.24s\n", scenario_name[i]);
                }
            }
        fclose(fp);
        }
    else
        {
        c_error(14,answer);
        goto input_x;
        }
    return;
    }
```