COMPUTERIZED SCHEDULING OF INTRAMURAL SPORTS

by

PAUL ALAN ROUSH

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July, 1982

Signature of Author . . . . . .
Department of Computer Science, July 30, 1982

Certified by . . . . . .
Thesis Supervisor

Accepted by. . . . . .
Chairman, Departmental Committee on Theses

TABLE OF CONTENTS

INTRODUCTION

M.I.T. has a reputation for having one of the nation's finest intramural sports programs. Those who have participated in the program probably need no convincing on this point. To those who have not, I would suggest that this well-deserved reputation stems from the fact that any student can play any sport at any level of competitiveness he desires. This requires first of all that programs be organized in all sports generating sufficient interest, and secondly that there be provision made for an unlimited number of teams in each sport.

Each sport is organized by a student who volunteers to be its manager. His first task is to draw up a schedule. In the more popular sports this can involve spreading upwards of 1000 games over many courts and a several month long season. At best this task is tedious and very time-consuming, and the prospect of having to deal with it scares away many potential managers. This has sometimes led to sports being cancelled for lack of a manager. In other cases, some hearty soul has undertaken the task only to find that the resulting schedule has enough mistakes in it to necessitate totally redoing it.

This thesis is an attempt to eliminate this major problem in an otherwise outstanding intramural program. The prospect of having a computer program to do the tedious part of the scheduling task, should encourage students to volunteer to be managers. Furthermore, this program is designed to produce a solution of higher quality than could generally be expected from hand scheduling. Some of the good features of this computer-generated solution include: games played by a single team are equally spaced throughout the season; each team plays the same number of

games; each team plays a variety of other teams throughout the season (i.e. there are no situations where two teams play each other over and over).

Additionally, this program allows the manager considerable power to "shape" the resulting schedule. It allows him to specify courts that must be used for certain leagues, or preferentially order the courts; to specify certain days of the week, or times of day that each league's games should be played on; and to specify certain days or certain times at which individual teams are unable to play. This last feature can be useful in avoiding conflicts between a house picnic and intramurals played by a team in that house, or it can also be used to avoid conflicts between games played by the same team in two different sports. Finally, the program is designed to allow the manager, if he so chooses, to "seed" the teams in each division. This could be done based on the previous season's records or however else the manager chooses. The seeding then results in the strongest teams playing each other toward the end of the season, with number 1 and 2 squaring off in the final week.

One additional feature that would be nice to add to the program would be the capability to reschedule rained-out games, leaving the rest of the schedule fixed. For the time being, though, the manager should be able to do this by hand without too much trouble. This point aside, the program should do most of what any manager would want it to, and be a valuable addition to the intramural program.

BACKGROUND

My interest in this project grew out of four years of very active participation in the intramural program. I played a number of sports each year (among them basketball, soccer, football, volleyball, water polo, ultimate frisbee, and softball) and drew from this experience a great respect for the diversity of the intramural program, numerous hours of enjoyment, and a good understanding of the strengths and weaknesses of the program. My knowledge of these strengths and weaknesses was furthered by serving two terms as my fraternity's athletic chairman, during which time I attended all of the Intramural Council meetings. The three major problems I noticed were difficulties in finding managers and referees for each sport and problems with the schedules that necessitated constant revising during the course of the season. This revising of schedules resulted in the further problem of teams showing up for games that didn't exist and failing to show up for games that did, because thay looked at an outdated schedule. Furthermore, the difficulty in finding managers stemmed largely from the fact that potential volunteers were scared off by the tedious nature of the scheduling task. Therefore, I felt that it would be a great boon to the intramural program to have a computerized scheduling routine that would make the manager's life easier and get the schedule right the first time.

The major problems I had seen with hand-scheduled seasons were in the form of teams left off the schedule entirely by the manager's oversight, teams who were only scheduled to play a few games (four for instance, when the rest of the teams were playing eight), and pairs of teams who wound up playing each other repeatedly rather than getting a

chance to play other teams in their division. The typical result of all of these problems was that the team(s) who had been slighted read over their copy of the schedule when they got it, noticed the problem (sometimes not until 2 or 3 weeks later), and notified the manager who then had to overhaul the schedule and send out new, revised copies. Since these revised schedules sometimes did not solve all the problems or else created new ones, more revisions were made and mass confusion generated. It was clear to me that a first schedule lacking these problems would be greatly appreciated by everyone involved in the sport. Another problem of a less serious, though still annoying, nature often cropped up in these schedules also. Quite often, teams would play 2 or 3 games in a week and then not play again for a month or two. I felt that a computerized scheduling routine could solve all of these problems and remove the major obstacle to finding volunteers to manage sports. Additionally, by making the initial scheduling task less time consuming and eliminating the need for constant revisions, it would free the manager up to spend more time looking for referees.

As I was thinking about undertaking the generation of such a program as the basis for my thesis, it came to my attention that Steve Pettinato had attacked the same project in 1980 for his bachelor's thesis, "A Sport Scheduling System". I read through his thesis and found that he had done a good job of stating the problem and the necessary features a program must have to produce a reasonable schedule. This high-level analysis proved useful to me as I decided on the constraints my algorithm should include and the basic manner in which it should function. If Steve Pettinato had been able to successfully implement his design, the need for my work in this area would have been minimal, though my algorithm

does expand on his in two significant respects (the capability to specify days and times that individual teams cannot play, and the capability to seed the teams). Unfortunately, Pettinato's code lacked many of the features that were deemed important in his high-level design. Among these shortcomings are the following.

First of all, the human interface was weak. This included both unclear prompt statements and tedious input of data. This second problem was quite serious since in several places the user was asked to supply a very large amount of redundant data. As a case in point, the manager is asked to give the times at which each time-slot starts once for each day of the season. Since these times should be the same for each day, the user is asked to do 150 times as much typing as necessary for a 150 day season. (i.e. if there are 8 time slots the user is asked to type in 1200 times rather than the 8 that are necessary.) Another problem along this line occurred when the code was unable to successfully schedule a division. In this case, all successful scheduling of previous divisions was lost, and the manager had to start over at square one, beginning with the re-typing of all the court availabilities.

Secondly, the data structures included in the program were insufficient to adequately describe the season. There was no facility included for changing court availabilities on the time-slot level, only on the day level. Thus if a varsity meet was to occupy one of the courts for only one hour, the manager would have to specify that court as being unavailable that entire day. Even more important than this, though, was the fact that time-slots available on a court on a given day were originally specified as a single contiguous block (i.e. the first and last slot numbers available on that day are requested). This leaves no provision for dealing with a court that is typically open in the morning, reserved for gym classes in the afternoon, and open again in the evening.

Finally, and most importantly, there is no provision made for shaping the resulting schedule through manager-defined constraints . The manager is not allowed to specify preferred days of the week or preferred times for scheduling games. While the manager may specify preferred courts, there is no weighting done -- he simply gives an order in which to search through the courts. This is somewhat helpful, but the code is structured so that finding the earliest available time slot is given higher priority. Thus, if all of the courts but the worst one were filled at 10:00, and all of the courts were empty for the rest of the day, the program would schedule the game at 10:00 on the worst court rather than at 11:00 on the best one. When a manager specifies court preferences this is not the sort of response he would generally desire. Additionally, the constraints of when each team is able to play are not addressed. The days and times during which each team is unable to play must be avoided when scheduling that team's games. Otherwise, the resulting schedule will be unusable until the manager does major revisions by hand.

PROBLEM STATEMENT

Flexibility Needed

In order for a computerized scheduling routine to be truly useful to the intramural program, it first of all must have a high degree of flexibility built in. It must be able to deal with sports like football that have four or five different levels of competition (A-league for near-varsity level play, D-league for those who don't even know the rules, B-league and C-league in between, and sometimes even E-league), less popular sports such as badminton and table tennis that have perhaps two or three leagues and 30-40 teams, and sports like basketball that have 150 teams wanting to play. Along with allowing for different numbers of leagues, it must allow for different numbers of divisions (each team plays others within its division during the regular season, and division winners meet in the playoffs) within each league and different numbers of teams in each division. The season may span many months or just one. Provision must be made for teams playing more games each in one sport than in another, and furthermore for the fact that often A-league teams will play more games than B- or C-league teams will in the same sport. There may be many courts or fields available or just a few, and these may each be available at different times of day and different days of the week. Further, the games may be scheduled only on weekends, only on weekdays, or throughout the week. Different times may be available for scheduling games from one sport to the next and even from one day to the next, and the number of games that can be squeezed into a given time period varies from one sport to the next as the lengths of the games are different. Allowance for specifying the values of all of these variables must be made if the routine is to be useful.

Control Over Scheduling Desired

Generally speaking, it is not sufficient to simply schedule enough games for each team on various courts at times those courts are available. There are normally a large number of criteria by which a manager judges one day and court and time-slot combination to be more suitable than another for scheduling a particular game. For instance, there are usually some courts or fields that are preferable to others and often only one of them is suitable for A-league play. Two cases in point are that all A-league basketball games are played in the Rockwell Cage and all A-league football games are played on the Rugby field. Sometimes the manager wants to schedule one league's game on a particular day of the week and/or a particular time, such as the traditional Sunday evening A-league basketball games.

The manager also takes into consideration days when a team is unable to play due to some outside conflict. This conflict might be in the form of a house picnic for the dormitory or fraternity the team is from or might be a game in a different sport that many of the team members will be participating in. As a logical extension of this concept I decided teams should be allowed to specify times on particular days that they can't play also, since they might, for instance, have an afternoon picnic and still be willing to play in the morning.

The manager also likes to have some control over the spacing between consecutive games of the same team. Generally speaking, it is desirable to have this spacing be uniform, and as large as possible given the restriction of fitting all of the team's games into the confines of the season (for a team playing G games in a season D days long, we would like

this spacing to approach $\frac{D}{G}$ days). As I noted in the background section, this is one area in which hand-made schedules are usually lacking, but that is no reason why a computerized system should not tackle the problem.

Another form of control not included when scheduling by hand that I felt would make a nice addition to this routine is the capability to seed teams within each division. Quite often, there are teams that are perennially strong in a given sport and the manager may have a feeling for the relative strength of various teams prior to the season's start. In this case, he might desire to input the seedings for these teams and have the schedule designed so that the strongest teams don't meet until the end of the season. A program giving the manager all of these forms of control over the way the games are scheduled would be a very useful tool.

Managers Not Computer-Oriented

While some intramural managers have considerable experience working with computers, others have never done any programming, and one cannot assume that such a manager will be overjoyed at the prospect of using a computerized scheduling routine. Therefore, if such a program is to be of any use it must be as "friendly" as possible to the user. This user friendliness clearly has to be the main goal in the construction of the program--- if the manager refuses to use the routine it is of no value at all. To achieve this aim, one thing that is clearly necessary is to provide very clear prompt statements that explicitly state the required input format. Additionally, each required data item should be prompted for separately rather than asking for a batch of data and expecting the manager to enter the right number of items in the right order.

One would also expect that any manager, whether he feels comfortable around computers or not, would not be particularly fond of the idea of spending numerous hours in front of a terminal entering data. Thus, it is important to write the code so that it requires as little input from the user as possible, and so that the amount of typing to enter any particular input is minimized. Finally, it is desirable to give the manager as good an overview of how to use the program as possible before he sits down at the terminal. This hopefully will put him more at ease, and through a little advanced planning allow him to minimize his data entry mistakes. To help in this respect a user's manual has been included as Appendix C.

Modifiability

One final facet of this problem is the need for a modular structure that will lend itself to future modifications. The previously stated needs for code that is flexible and offers the manager a great deal of control over the scheduling constraints necessitate a long program. Basic principles of good programming call for any large program to be developed modularly, and without this modular structure the initial debugging task might have been virtually impossible. In this case, there are also other good reasons for insisting on a modular structure. Since the final goal is to provide a service to the intramural program, the design of the code should take into account the likelihood that someone will in the future wish to either tinker with the constraints used or add new ones to expand on the routine's usefulness. In order to make such modifications possible it is necessary to write the code in a highly modular fashion.

GENERAL ALGORITHM

The first step in deciding on the basic design of the algorithm was determining what portion of the scheduling problem it should attack. At one extreme, it could take on every conceivable scheduling scenario and require the manager only to enter the initial constraint input and any subsequent considerations that arise and call for modifications, such as rained out games that need to be rescheduled. On the other extreme it could be a very simple routine that ignores many of the more subtle constraints and produces a first-draft schedule that the manager must then heavily modify. This would presumably still be easier than starting from scratch, but as long as we're going to use a computer why not have it do as much of the work as possible? I therefore decided to include in my algorithm the ability to provide all of the flexibility and control described in the first two parts of the Problem Statement section.

There were two features which I considered including but decided against. The first was allowing for the scheduling of several sports at once, and cross-checking to avoid conflicts such as teams from the same house having football and soccer games scheduled at the same time. This is traditionally a problem, but I did not feel that parallel scheduling of several sports was the best answer for the following reasons. First, it would require a great deal of coordination between the managers of the sports involved, since the scheduling would need be done at the same time for each sport (currently, sports played in the same season are scheduled at times weeks apart from one another -- the particular time depends on the manager's whimsy) and in order to cross-check for conflicts the team listings would have to reflect which house the team is from and what other

teams in other sports also contain members of that team. This necessity of coordinating with other managers would add to each manager's headaches, when the purpose of this program is to make their lives easier. Secondly, these additional headaches would be bought at the expense of code that is more complicated, and therefore more difficult to modify and more expensive to run. Finally, and most importantly, there is a simpler solution. Since provision is made for each team to specify days and times on which they cannot play, they simply note those times they are scheduled for games in one sport, and request not to be scheduled at those times in the next sport. This has the pleasant quality of diverting some of the resposibility for developing schedule constraints from the manager to the individual teams, thereby spreading out the work load.

The second feature I decided against including was a capability to reschedule rained out games without modifying the rest of the schedule. This would be a valuable feature and probably should be added at some point. My decision to leave this out was based on two observations. First of all, it would add some complications to an already complex task. For this reason, I felt a more suitable approach would be to first get a basic scheduling routine working, and then add in rain game rescheduling at a late point. Such adding on of features often causes problems as the additional code does not mesh well with the original algorithm. In this case, however, the feature to be tacked on is sufficiently disjoint from the main algorithm that there is little need for smooth integration. Secondly, my feeling was that leaving this feature out would not greatly lessen the usefulness of the routine. While the initial scheduling task is tremendously difficult to do by hand, given an initial computer-generated schedule, the manager should have no trouble rescheduling a few individual rained-out games.

After deciding on the scope of the algorithm, the next step was to determine just how to give the user the sort of control discussed in the problem statement section. To govern the scheduling of each individual game, a set of constraints were developed. I decided that some of these constraints should be mandatory and the others should be weighted according to their degree of desirability. I will give an overview here of what sorts of mandatory and preferential constraints were included. For a more detailed description of how the constraints were implemented, how the preferential constraints were weighted, and how the set of constraints were combined to choose the best spot in the schedule to place each game, see Appendix A.

Mandatory Constraints

The first set of mandatory constraints are simply the days and times on which each court is available. To minimize the amount of input the manager must provide, these are entered in the following manner. First the manager specifies the number of courts and whether the sport will have games scheduled on weekends, weekdays, or the full week. He then specifies what time-slots are typically available for each court on each day of the week. This sample week is then replicated through the length of the season. The bounds of the season are specified by typing in the months that it spans, the number of days in each and what day of the week each month starts on. To deal with the fact that the first and last months are probably not fully included in the season, the season's first and last dates are also asked for. Once this basic calendar has been created, the manager is asked for exceptions to it in the form of full days that are unavailable (to take care of holidays and such), and

times that are unavailable on particular days (this might result from a varsity contest being played on that court that evening). Provision is also made for the addition of extra time slots that are not typically available.

In addition to the court availability constraints, there are also mandatory timing constraints on a team by team basis. As each division is scheduled, the manager is asked for full days and blocks of time slots on particular days on which any of the teams in that division cannot play.

The final mandatory constraint provides for the specification of a court or set of courts that must be used for games played in a particular league (this generally is only used for A-league games).

The actual scheduling is done on a division basis. That is to say, all of the games to be played by teams in one division are scheduled and then the program moves on to the next division. All of the divisions in one league will be scheduled and then the routine will move on to the next league and loop through its divisions, scheduling each of them. The mandatory court constraints are entered at the league level, and than all of the divisions within that league are scheduled using those constraints. The team by team time constraints are entered at the division level and are reset as the program moves on to the next division. The court availability constraints are entered outside of the league loop, as they pertain to the entire schedule.

As the program attempts to schedule each game it looks first at those mandatory constraints to determine what time slots on each court on each day are feasible places to position that game. (In actuality it is much more selective than this -- it does not examine all combinations. A description of which slots are checked for feasibility is given in the

"Search" section later in this chapter.) Once the program has come up with a group of feasible time slots it examines each of them to determine how well they meet the various preferential constraints and chooses the one with the best overall "figure of merit."

Preferential Constraints

To allow for subtle shaping of the schedule, there are three areas in which the manager may specify preferential constraints. These are preferred courts, preferred days of the week, and preferred time-slots. These items were designated as desirable rather than mandatory since, generally speaking, it is not as crucial to satisfy them as to satisfy the constraints that were designated mandatory. There is, nevertheless, provision made for turning any of these constraints into what are essentially mandatory requirements by simply giving them a high enough weight. These constraints will now be discussed one by one with a more detailed description appearing in Appendix A.

When the manager first enters the court names he is asked to give them in their default preferential order. No explicit weighting is given at this point, but this gives the program the order in which to search the courts in the absence of tighter constraints. Should the manager wish to change this default ordering for a particular league or heavily favor one or more of the courts over the others, he can do this when he is specifying constraints for the league. Setting the weights extremely low will have the effect of simply changing the default preference order. Setting a court's weight very high will essentially make it mandatory, though explicitly doing so is a better idea. The reason that explicitly designating the court as mandatory is better is that in that case if there is no opening on the given court the

manager will be notified of the problem. This preferential court ordering is only foolowed, of course, if no mandatory courts are specified for the league being scheduled.

Occasionally the manager will want to schedule games in one league on a particular day of the week, such as the Sunday night A-league basketball games I mentioned before. To allow for this he is given the options of specifying both preferred days of the week and preferred timeslots for each league. The weighting given to the constraint preferences is in the form of the number of days the manager is willing to postpone the scheduling of a game in order to satisfy the given criterion. The result, then, of specifying a weight of 10 for scheduling on a Sunday, is that if an open slot is found on Friday the 15th the program will continue searching until the 25th in hopes of finding an open slot that falls on a Sunday instead.

Search Strategy

As I have previously stated, the scheduling is done on a division by division basis. The first thing that is done for each division is the generation of an array of team pairings. Basically this pairings array can be thought of as containing N blocks, where N is the number of games that each team is to play. Within a block each team in the division plays one and only one game. (Actually this is slightly oversimplified -- for a fuller description see the write-up on the TEAM_PAIR module in the "Code Structure" section.) Since the team pairing arrays are designed to have teams play a different opponent each week, a given team may play at the end of one block and the start of the next. This being the case, to meet the goal of equal spacing between a particular team's games it

is necessary that the blocks be packed tightly, and that the spacing between blocks be as wide as possible. As an example of what might happen otherwise, consider a 3 game season with 2 week long blocks that have no space between them. In this case, one team might play at the end of the first and third blocks and start of the second. This team's schedule would then consist of games on back-to-back days followed by a month long layoff prior to their final game. Providing widely separated, tightly packed blocks of games is therefore a primary goal in searching for appropriate times to schedule each game.

One of my concerns in writing this program was to design it in such a way that it would not cost too much to run. In a sport like basketball there can be as many as 150 teams playing 8 games each, giving a total of 600 games to be scheduled. In general, there are more time slots made available than the number of games called for. This padding is necessary to allow for rescheduling rained out games, and without it the last few divisions to be scheduled would probably wind up a mess since there would be essentially no choices about where to schedule games. Even if there were only as many time slots as games, though, in a 600 game season there would be 600!, or upwards of 10^{1500}, possible schedules. By breaking the teams up into 4 to 8 team divisions and scheduling on a division by division basis, the number of possible arrangements is greatly decreased, but is still on the order of 10^{50}. Clearly then, it is not feasible to search through all possible schedules for the best one. The routine must methodically eliminate all but a very small fraction of the possibilities and then evaluate the merit of just these few. It is clearly quite possible that the optimal solution will be among the 99+% that were never even evaluated. By being very intelligent about which few

combinations it decides to look at, the program can nevertheless be relatively certain of finding a solution that is quite good. The bottom line is that the algorithm is designed so that it settles for a very good solution rather than looking for a perfect one.

In an attempt to minimize the number of days searched while producing a schedule having widely separated, tightly packed blocks, I decided on the following approach. The manager would designate a minimum spacing between blocks, and then to schedule each block of games in the division, the program would search only those days which were at least that many days past the last game in the previous block. While in some cases this might result in overlooking an excellent choice only one or two days earlier, it does allow for a significant reduction in searching and insures that blocks do not get too narrow a spacing between them.
From this starting point, the routine searches forward only, and generally searches for a very limited distance. This is clearly good from the standpoint of minimizing execution cost, but one might have doubts about how good a solution such a limited search could make. However, from the standpoint of providing equal spacing between a team's games the ideal positioning for the game is right at the point where that search starts so there is no point in checking days far removed from that spot. In the absence of preferential constraints the program will in fact choose the first open slot it finds and schedule the game there. When preferential constraints are present, the program will search farther in an attempt to satisfy the given constraints. How much farther is determined by how heavy a weight the constraints are given. When this longer preference-seeking search is taking place there is clearly no need to look for open slots on a day that does not satisfy the constraints in question. There-

fore, a preliminary scan of each day is made to determine whether it even has the potential to improve on the best-fit found prior to that day. A flowchart diagramming this search process can be found in the next chapter, in the section discussing the DIV_SCHED routine. For a more detailed description of how the preferential constraint weights alter the length of the search, see Appendix A.

CODE DESCRIPTION

In this chapter I will discuss the actual code I wrote to implement the algorithm described previously. The program was written in PL/1 and run on M.I.T.'s Multics system. PL/1 was chosen because it is the language I was most familiar with that had the necessary flexibility of data structuring and bit-handling capabilities. It was important to be able to describe available time-slots on the court and team levels with single bits in order to keep the memory requirement reasonable. As a side note, it should be mentioned that I refreshed my memory of PL/1 by taking a look through Appendix B of Introduction to Structured Programming Using PL/1 and SP/k by Conway, Gries and Wortman. This appendix summarizes PL/1 and might be helpful to any reader wanting to brush up on the language.

One of the goals mentioned on the Problem Statement chapter was to keep the code highly modularized. To this end, the code was broken up into ten modules -- eight subroutines and two functions. The functioning of each of these modules will be described later in this chapter. An overview of the interaction between them, in terms of flow of control and passing of data arrays, can be seen from the figure at the start of that section (Routines section). I will now discuss the major data structures used in the program. Some data structures that are important to the functioning of specific routines but which are not crucial to an understanding of the program as a whole will be covered in the section discussing the appropriate routine.

Data Structures

The format and function of each of the major data items follows. Several of these are PL/1 structures with formats too involved to reproduce here. In those cases, a mention is made of what portion of the code the reader can find the appropriate declaration in. Since the data discussed in this section is that which is most important to the functioning of the overall code, most of the items are accessible throughout the program. For those items whose scope is not the entire program, explicit mention is given of the outermost block in their scope.

Just as I felt it inappropriate to include complex structure declarations in this section, I also did not desire to write these declarations over in each routine that was to access them. I especially did not want to include their format in the argument declarations for the subroutines they were to be passed to. For this reason, structures that needed to be widely accessed were declared in "include" files and were allocated space based on pointers. These pointers were then passed in argument lists rather than passing the structure itself.

MISC is a structure whose declaration can be found in the STRUCTURES. INCL include file. It contains a variety of miscellaneous data such as sport-name, number of leagues, the names of each league and the number of divisions in each, the number of courts and their names, and the names of and number of days in each month. Most of this data is entered in the MAIN routine, with the exception of the court data which is entered in the COURT_AVAIL routine.

SEASON is another structure whose declaration can be found in the STRUCTURES include file. It is filled in the CREATE_SEASON routine, holds the number of days in the season, and correlates each season day to the

actual calendar. This correlation includes the day of the week, month and date of each day in the season. The day of the week is stored as a number (Sunday= 1, Saturday= 7, etc.) both because this saves space, and because this format is more easily used in most parts of the program. Thus if we wanted to know whether the 12th day of the season fell on a Tuesday, we would check to see if SEASON.DAY(12). DAY_OF_WK was equal to 3. The REL_DATE entry gives the number of actual calendar days from the start of the season to the day in question. This is used in the search portion of the scheduling process to give a measure of the actual number of days from the start of the block to the day being currently checked. The DATE entry could be used for this were it not for the fact that searches across month boundaries would then require much unnecessary calculation. The need for the AVAIL entry can be seen in the discussion of the CREATE_SEASON routine --basically it notes days that would normally be part of the season but are unavailable (holidays for instance) by setting this bit to Ø.

FACAVAIL is a 150 x 10 array of 15-bit entries. The first index specifies a day by giving its index in the SEASON array. The second index specifies the court number. The bit string for any day & court combination gives its availability during each of the time slots. Thus, a 1 in the 3rd bit of FACAVAIL(110, 5) means that the 5th court is available during the 3rd time slot on the 110th day of the season.

DIVISION is a structure whose declaration can be found in the DIV_DCL include file. Its scope is bounded by the DIV_SCHED routine, and on each call to DIV_SCHED is filled with data pertaining to that particular division. This data includes the number of terms in the division, the number of games each will play, and a team-number that is unique to that team through the life of the run. This team number is used to uniquely identify

each team in the overall schedule listing by its position in the scheduling order (i.e. the 34th team scheduled is 34).

PAIRINGS is a 14 x 4 x 2 array of FIXED values whose scope is bounded by the DIV_SCHED routine. It is filled in the TEAM_PAIR routine with the teams that will meet in each game scheduled for that division. The first two indices correspond to block number, (if each team plays 6 games there will be 6 blocks), and game number within that block, respectively. The third index is either 1 or 2 depending on whether you want to access the first or second of the teams paired off in that game. The value of each entry is the team number. This team number is not the same as that contained within DIVISION, however, this one running from 1 to 8 in an 8 team division. PAIRINGS(4,3,1) and PAIRINGS(4,3,2) contain the team numbers of the two teams meeting in game 3 of the 4th block of games. These team numbers represent the index of the team within the division. From this division-oriented team number (call it TN) we can obtain the sport-wide number for the time by taking DIVISION.TEAM_NUMBER(TN).

TEAM is a structure whose declaration can be found in DIV_SCHED and whose scope is bounded by the same. It is filled at the beginning of the DIV_SCHED routine with the time constraints for each team in that division. For each day in the season and each team in the division, TEAM contains a single bit telling whether that team can play on that day, and a bit-string telling which of the time slots on that day are acceptable to the team. By putting TEAM in this bit-matrix format (as opposed to a list of problem times) it became possible to overlay sections of FACAVAIL and TEAM to quickly determine (using bit-wise ands) just which slots were available to schedule a particular game.

LEAGUE is a structure containing constraints that apply to the league being schedule at that time. Its 'declaration' can be found in LEAGUE_DCL. INCL and it is filled with data in the SCHEDULE routine as processing starts for each new league. It contains information on the mandatory or preferential courts designated for the league, preferred days of the week and time-slots, and the minimum number of search-days called for by the weighting of the given constraints (this is explained in Appendix A). The preferred time slots are recorded as a list of contiguous blocks of slots each associated with a particular day of the week. This allows, for example, the specifying of Sunday evening (from 6-10 pm. perhaps) times as being preferable for A-league basketball games.

SCHED is the structure that keeps track of the games that have been scheduled -- the teams playing, and the day, court, and time-slot associated with the game. Its declaration can be found in STRUCTURES.INCL and it is filled in the DIV SCHED routine. Its format is that of an array indexed by (day-of-the-season, court-number, time-slot-number). Each element thus indexed is a pair of fixed values giving the team numbers of the combatants scheduled in that slot, or zeroes if the slot is still empty (The team-numbers used are the sport-wide ones mentioned in the discussion of TEAM on the previous page). An alternative format was considered that would list games in the order scheduled and would contain day, court, time, and team numbers in each entry. Since this array has the potential to grow very large, it would be nice to minimize its size to whatever extent is feasible. A list of games could require considerably less storage (depending on the percentage of time-slots that will eventually be filled) since it does not have to store zeroes for every slot that could have been used but wasn't. The chosen format, however, is preferable for two reasons.

First, it is desirable to have those zeroes, since they allow the manager to look at the schedule printout, and at a glance tell where there are empty slots still available. (These can then be used if he needs to reschedule a rained out game.) Secondly, the printout of the schedule needs to be done in chronological order to be of any use, and that is not the order in which the games are scheduled (i.e. the program moves through the season scheduling division 1, then returns to the start of the season to begin scheduling division 2, etc.) Thus, if a list format were used, each time a printing was to be done this list would need to be sorted by day, court, and time. It seemed to make more sense to put it in this format to begin with. However, the maximum number of season-days, courts, and time-slots that I wanted to make allowance for were 150, 10, and 15 respectively. Since two fixed numbers comprised each entry this called for a 45K array, which was much larger than I wanted to have to allocate space for. In almost all cases, however, the actual number of days, courts, and time-slots will be well below these maximun values. I therefore decided that the array should be dynamically allocated at just the right size after these actual values were known. There was initially a problem with this approach since I had planned to have the MAIN routine call DIV_SCHED directly. The SCHED array could not be declared in the MAIN routine since the needed values were not known on its entry. The array could not be declared inside DIV_SCHED or it would lose track of the data for the past division each time the routine was called to schedule the next one. I thus decided that the best solution was to add a shell routine around DIV_SCHED called SCHEDULE whose only purposes would be to initialize the SCHED array, input constraints for each league and call DIV_SCHED for each division.

Routines

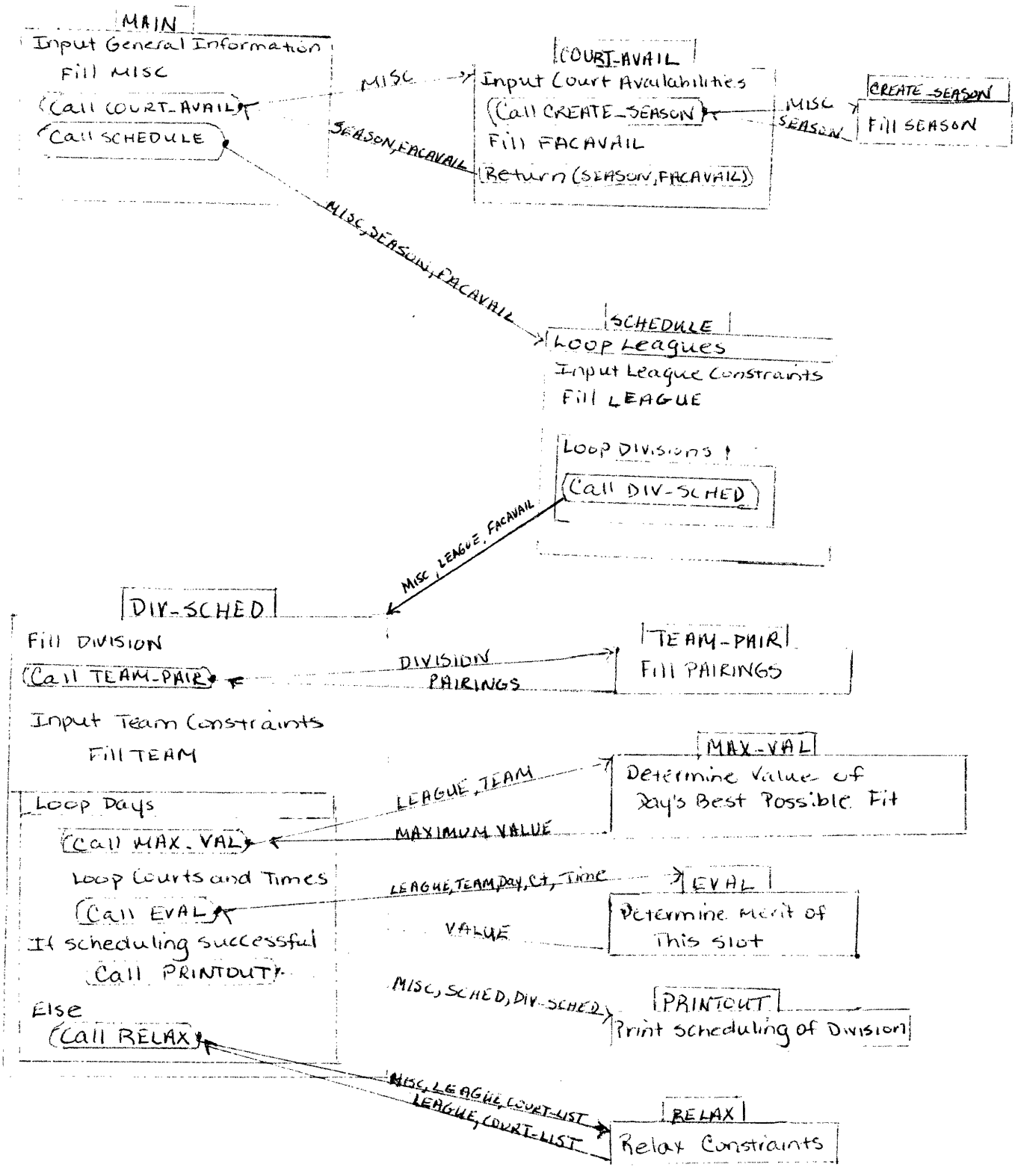
This completes the discussion of data structures that are used widely throughout the program. I will now discuss the subroutines that comprise the program, with the exception of MAX_VAL and EVAL which are discussed in Appendix A. For each routine, I will discuss its function in broad terms, how it achieves its purpose, subroutines it accesses, data flow to and from it, and data structures that are important to its functioning but were not used widely enough to be mentioned in the previous section. An overview of the manner in which these routines interact, in terms of both data flow and flow of control, can be seen in the figure on the following page. Listings of the actual code can be found in Appendix B.

MAIN

The MAIN routine is the starting point for the program. Its task is limited to requesting the input of a variety of information that will be needed throughout the entire program, and calling two subroutines that then see that all the actual work gets done. There is no computation done in this routine. The two subroutines called are COURT_AVAIL, which determines the availabilities of the various facilities throughout the season, and SCHEDULE, which loops through the leagues and divisions and sees that each one is scheduled in turn. These subroutines access a number of other subroutines to help with various parts of their tasks.

The first task of the routine is to input the information that is needed to fill the MISC data structure. Since each input has been prompted clearly in the interests of making the program as easy to use as possible, the functioning of this portion of code should be easily understandable by looking at the listing. The three terms used to describe the sport-week

INTERACTION OF MODULES



(misc. sport_wk.SW1. and SW2, and SW3) represent the first day in the week, the last day, and the increment between consecutive days for this sport. Thus, these terms can be used as the "from", "to", and "by" indices of a do_loop designed to step through the days in a single week of the season. The barest minimum of data regarding the days spanned by the season is requested -- namely the number of months, number of days in each, and the day of the week that each starts on. From this information combined with the sport-week data, a full season calendar can be constructed by the CREATE_SEASON routine. This is one example of the efforts made to keep user input to the absolute minimum required.

Once the MISC structure has been filled with the necessary data, that data is passed to the COURT_AVAIL routine which determines what times each court is available on each day of the season and returns this information in the FACAVAIL array. In addition, COURT_AVAIL calls CREATE_SEASON to produce the season's calendar as mentioned above, and then returns this information to the main routine in the SEASON data structure.

SCHEDULE is now called with pointers to the MISC and SEASON structures, and using this data, the routine sees to the actual scheduling.

COURT_AVAIL

COURT_AVAIL is a subroutine called by MAIN to determine the times at which each facility is open on each day in the season. The routine is passed the data in the MISC array and returns the availability information in the FACAVAIL array. Additionally, the routine calls CREATE_SEASON to expand the limited season-size information in MISC out into a full calendar which is then used in the filling of FACAVAIL and subsequently passed back to the MAIN routine.

The routine starts off by requesting the number of courts and the names of each. It then determines from the time-slot data in MISC (starting time of first slot, length of slots, and number of slots) the time each of the slots starts and prints these times so that the user can in the future refer to time-slots by their index (i.e. first slot has index 1). The manager is then asked to input the times each court is available during a typical week. To minimize typing this data is entered in the form of n-bit strings (where n is the total number of slots in a day) with 1's corresponding to available slots and 0's to unavailable ones. Once the sample week is complete, the user is asked to verify that the input was correctly entered (since entering a number of bit-strings can easily result in typos). If the input is verified, the sample week is replicated through the season and the result is the FACAVAIL array. If not, the user can specify which bit-strings were incorrectly entered and change them. When he is satisfied that they are all correct the replication is done.

Once the sample week has been replicated, exceptions to this "ideal" season are entered. The user is asked for the court whose availability is to be changed and then the month and date to be modified. The new availability string is then entered and the user is asked if there are more changes to be made. When no more exceptions exist, the FACAVAIL array contains the precise court availabilities for the entire season, and this data is returned to the MAIN routine so that scheduling can begin.

Throughout this routine, count is kept of the total number of slots available during the season on each court. The number of slots open on court I is MISC.COURT(I).TOTAL_SLOTS. This information is updated accordingly as scheduling proceeds, so that when a division cannot be successfully scheduled, the user can check to see whether the problem was a lack

of open slots on a mandated court, or lack of slots overall, or whether this had nothing to do with the trouble.

CREATE_SEASON

CREATE_SEASON is called by COURT_AVAIL to generate the season's calendar. It is passed the data in MISC and returns the completed calendar in the SEASON array. This "calendar" is actually in the form of a list of days in the season. Each of these entries is comprised of the day of the week on which the day falls, its month, its date, and its "rel_date" which is a measure of the number of real (rather than seasonal) days between the start of the season and the day in question. This last quantity is used to determine whether a search to satisfy a preferential constraint should be carried farther than the season day currently being looked at (see Appendix A for more details of how this is used).

The routine creates the season by replicating the sport_week (the days SW1 to SW2 by SW3) found in MISC through all of the months in the season, from the FIRST_DATE to the END_DATE. The first step in scheduling each month is to determine the correspondence between dates and days of the week. This is accomplished by using the day of the week the month starts on to determine the date of the Sunday that starts that month's first week (SUNDATE). This SUNDATE is generally negative since the 1st might fall on a Wednesday for instance, and the Sunday beginning that week is 3 days earlier. The "date" at the start of this week now known, the routine loops through the sport-week incrementing the date as needed until the first day of the month is found (or in the case of the first month, the FIRST_DATE in the season). Following this point, the day-of-week, month, date, and rel_date are recorded for each day. For the rest of the weeks

in the month, each day in the sport-week is entered into SEASON until the end of the month (MISC.MONTH(I).LAST_DATE) is encountered. At this point the process repeats for each month until the full season is created. The days at the end of the last month that fall past END_DATE are then trimmed off. Since provision needs to be made for dates on which major activities or holidays will cause a day in the sport-week to be removed from the schedule, there follows a section to input exceptions. The days to be stricken from the season are noted by setting their AVAIL bit to "0".

SCHEDULE

SCHEDULE is called by MAIN with the MISC and SEASON structures and FACAVAIL as input. SCHEDULE allocates sufficient space for the SCHED structure (which will hold the actual schedule) based on the number of days in the season, the number of courts, and the number of time-slots. The fact that this structure is dynamically allocated with these actual values is important since simply declaring it with all three indices set to their maximum size would result in a huge array. SCHED is then zeroed out and the leagues and divisions are looped through, calling DIV_SCHED to schedule each division. At the start of the league loop, however, the mandatory and preferential constraints for that league are entered and these are placed in LEAGUE.

DIV_SCHED

DIV_SCHED is the longest of all the routines and the true heart of the program. It sees to the actual scheduling of each division, allows for appropriate corrective measures to be taken when a division cannot be scheduled, and provides printout of the results when they can. In performing

these tasks it calls on two functions; MAX_VAL and EVAL, and three subroutines; TEAM_PAIR, RELAX, and PRINTOUT. It receives the data in MISC, and SEASON, and the constraints in LEAGUE and FACAVAIL as input, and updates SCHED as each new division is scheduled. There are two data structures not previously described that figure importantly in this routine -- DIVSCHED and CHANGE. The declaration of DIVSCHED can be found in the DIV_DCL include file, that of CHANGE at the start of this routine.

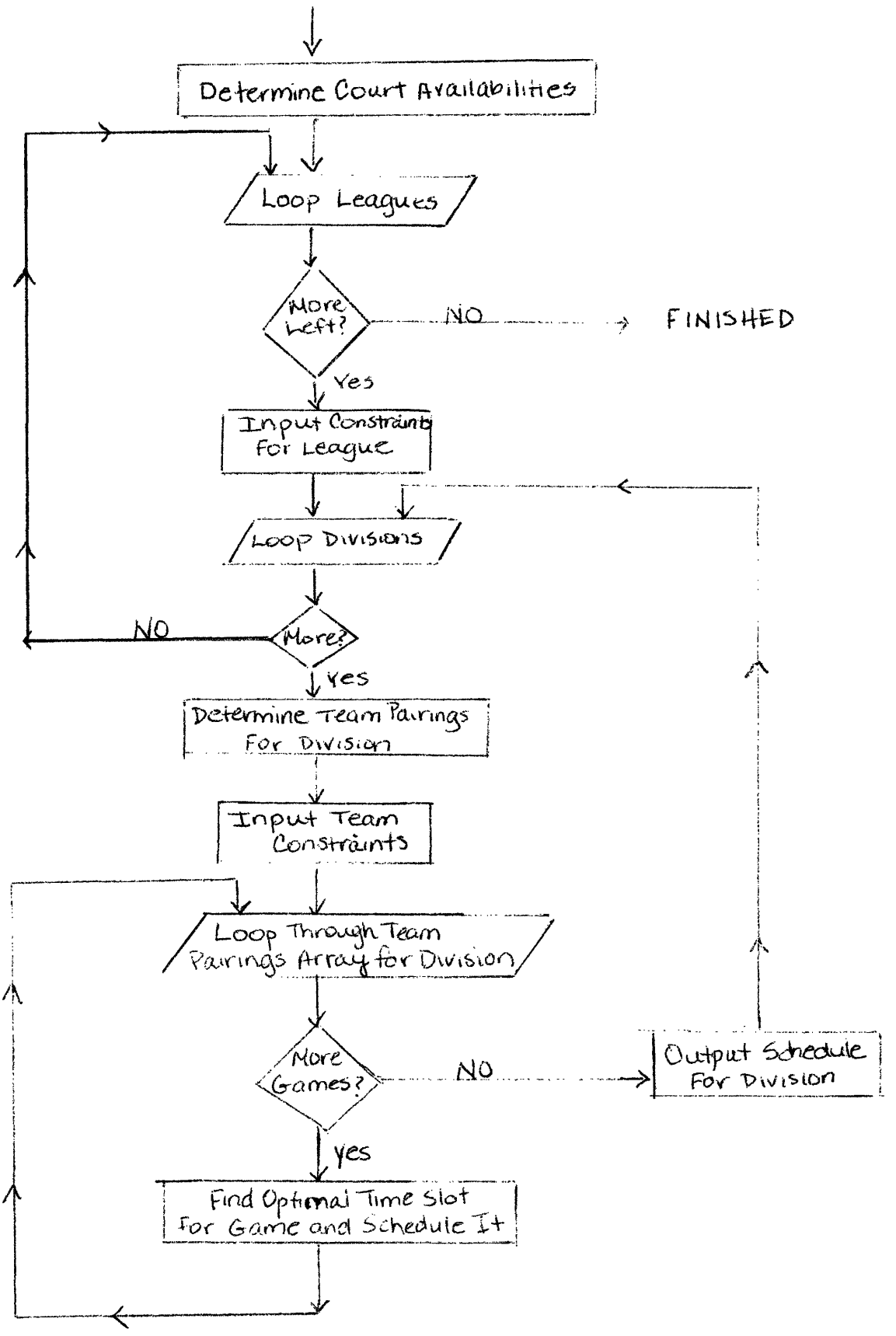
DIVSCHED is a list used to keep track of the games that have been scheduled for the current division. It is used by the PRINTOUT routine when it is desired to see the schedule for just the division most recently completed. When the user wants to see the full schedule, the PRINTOUT routine uses the SCHED array instead.

CHANGE is a list of changes that have been made to the SCHED array during the current call to DIV_SCHED. This is used to restore SCHED to its previous state in those cases where the division has been partially scheduled (and the SCHED array therefore modified) but cannot be successfully completed. If the user decides to give up in such a case there is no problem. If he wishes, however, to relax some constraints and then try again to schedule the division, SCHED must first be restored to the way it looked on entry.

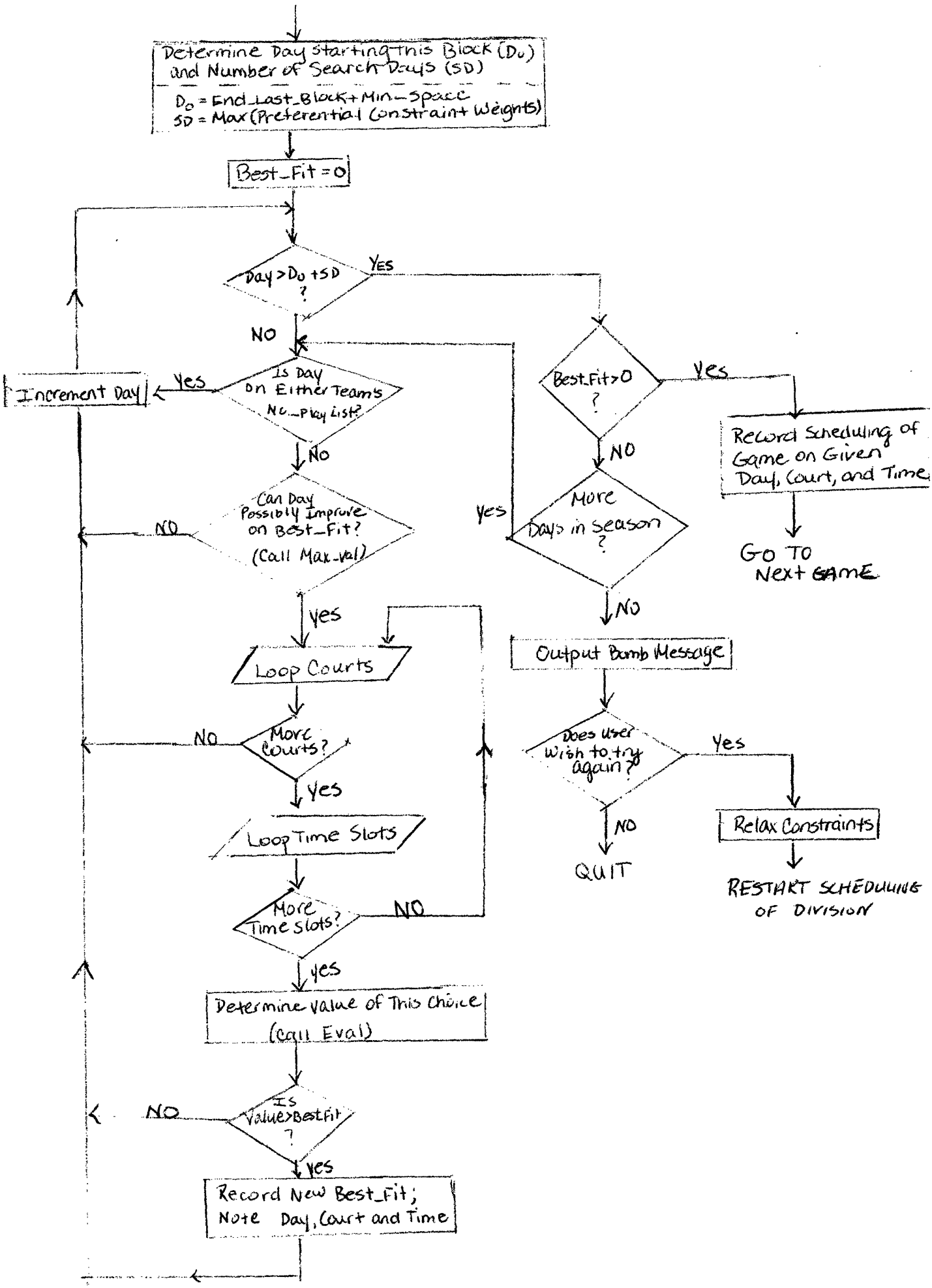
On the next two pages can be found flow charts diagramming the basic scheduling procedure and the method used to select the optimal time slot for the placement of a particular game. Referring to these may be helpful in understanding the explanation that follows, as this routine is complex.

The routine begins by inputting the number of teams in the division, the names of each, and the number of games that each team should play. The total number of games that will need to be scheduled for the division is

OVERVIEW OF SCHEDULING PROCESS



CHOOSING OPTIMAL TIME SLOT



then calculated (DIVSCHED.N_GAMES). In the case where the division has an even number of teams this is trivial -- it is simply (# of teams) x (# of games per team)/2. However, in divisions with an odd number of teams it is more difficult to determine just how many games must be scheduled to insure that everyone plays at least the minimum number specified.

The TEAM_PAIR routine is then called with the number of teams and the number of games each will play and returns the PAIRINGS array filled in a seeded fashion (teams 1 and 2 always play on the last week, for instance). The constraints for each team as to what days and times they cannot play are then entered, and these values are placed in TEAM.

The next order of business is to create a list of courts (COURT_LIST) designating those that will be searched as this division is scheduled. In the case where the current division is in a league with one or more courts that have been designated as mandatory, these will be the only courts on the list. In the case where no mandatory courts are specified, the preferential court ordering for that league (if any was given) will be used, or as a final option the order in which the court names were entered will be used as the default ordering.

At this point the actual scheduling is ready to begin, and the minimum spacing (MIN_SPACE) between blocks of games is asked for. In a division containing 6 teams, a block would consist of 3 games (at the end of the block each team has played once). The blocks are looped through, and within that loop the games within the block are looped. The teams to meet in each game are found by observing the entry in PAIRINGS specified by the current block and game numbers.

With a particular game to schedule, the routine now moves through the days of the season looking for open time-slot & court pairs on that day that

satisfy as many preferential constraints as possible. As each new day is examined, it is first checked to insure that the end of the season has not been reached. If it has, and an opening has been located then the game is scheduled in that opening. If no opening has been found, the routine prints an error message, allows the user to incrementally request possibly useful diagnostic information, and then, if he desires, relax some of the constraints and try once more to schedule the division. This diagnosis and constraint relaxing is done by the RELAX routine.

If the new day being examined is part of the season, a check is then made to see whether it is at least SEARCH_DAYS past the start of the block. If it is, the routine can be assured that if any opening has been found it is, in fact, the best-fit (for an explanation of this see Appendix A), and it schedules the current best-fit. In the case where no opening at all has been located the program will continue searching until an opening has been found or the end of the season reached.

If the routine is still interested in exploring the current day after these preliminary checks, it calls MAX_VAL to determine whether there is any hope that the day might improve on the current best-fit. Essentially, MAX_VAL determines the best value of any of the slots on that day without worrying about whether the slot is open.

If MAX_VAL indicates the search should proceed further, the courts in COURT_LIST are looped through. If there are preferred time slots these are looped through before the rest of the slots, but in any case, all slots will be examined (at least until an open one is found). The availability of the time-slot being examined is determined by the "anding" of both team's availabilities on that date with the FACAVAIL entry corresponding to that court and date. When an open time slot is seen, EVAL is called to deter-

mine how many preferential constraints it satisfies and how important they are. If the "value" of this slot, court, and date combination is higher than the best-fit previously found, this new best-fit is recorded. Since the preferred time-slots are searched first, there is no chance that another time found on that day will produce a higher value, so the routine moves on to the next day in the season.

TEAM_PAIR

TEAM_PAIR is called by DIV_SCHED with the number of teams in the division and the minimum number of games each is to play. It then fills the PAIRINGS array with the necessary number of games and passes it back to DIV_SCHED. The match-ups it creates are designed such that the variety of opponents for each team is maximized and the strongest teams (if the teams are entered in seeded order) will meet at the end of the season. For example, in a 4, 6, 8-team division, teams 1 and 2 will meet in the last week, as will teams 3 and 4, and teams 1 and 3 will square off during the next to last week.

The basic data used in the creation of the PAIRINGS array consists of a set of 5 arrays -- one for use with 4-team divisions, one for use with 5-team divisions, etc. Each of these arrays has just enough blocks of games in it so that each team will play every other team in the division exactly once. The matchups included in these arrays were designed to allow for seeding, with the stipulation that the last block in the data array should always be used to fill the last block to be included in the PAIRINGS array. To accomplish this, the number of games desired per team is first used to calculate the number of blocks needed (for a division having an even number of teams there is a one-to-one correspondence; for a division

having an odd number of teams one to two additional blocks are required). The number of blocks needed (n) is then compared to the number of blocks in the basic data array (m) and one of the following is done. If m= n, then the first block of the data array is placed in the first block of the pairings array, the second in the second, etc. If m>n, (m= n + i), then the first i lines of the data array are unused, the (i + 1)st is placed in the first block of PAIRINGS, etc. If n>m, then enough lines of the data array are skipped so that after the remaining ones are used, there will be just as many blocks left to fill in the PAIRINGS array as there are entries in the data array (i.e. if n= m + i, [m - i] lines will be skipped, the last i blocks in the data array will be used to fill the first i blocks in PAIRINGS, and then all m blocks in the data array will be used to fill the last m blocks in PAIRINGS).

RELAX

The RELAX subroutine is called by DIV_SCHED when it is unable to schedule a division and the user wants to relax some of the constraints and try again. The routine is passed the MISC and LEAGUE data structures, as well as the COURT_LIST array used to specify the courts to be searched and the order in which they will be scanned. Currently, the routine has a limited range of options, but the module could easily be expanded to do more. Part of the reason for its current limited scope is that in the vast majority of the cases, the only constraint that the manager will want to relax will be the minimum spacing between blocks. Since this constraint is set in DIV_SCHED right at the point where the control is returned to when rescheduling begins, there is little reason to include in RELAX the capability to modify it.

The routine as it is currently structured allows the user to either eliminate the mandatory courts for the league being scheduled, or add to the number of mandatory courts. Either approach will improve the chances of successfully scheduling the division, but neither one will necessarily do the trick if, for instance, the real problem stems from too large a value for MIN_SPACE. If the user chooses to eliminate the mandatory court list, he is then given the opportunity to replace this with a preferential court list, giving a weight to each of the courts. The COURT_LIST array is appropriately updated and returned to DIV_SCHED since the courts actually searched through are determined by its contents.

PRINTOUT

The PRINTOUT procedure is called by DIV_SCHED following the successfull scheduling of a division. It is passed the MISC, SEASON, and DIVISION structures to provide it with the information it will need for printing clear output (such as month names, court names, etc.). SCHED and DIVSCHED are passed in to provide the actual scheduling data for the sport as a whole, and for the division that was just processed. Finally, the league and division indices for the just completed division are passed in to allow the proper information to be accessed out of MISC.

The routine starts by printing the names of each team in the division and their sport-wide identifying numbers. Next, a chronological listing of the games scheduled for this division is printed. This listing provides the team-numbers of the combatants; the time, day of week, month, and date that the game is scheduled for; and the court on which the game will be played.

The routine then asks the manager if he would like to have schedule

listings printed by team. If he asks for this, a listing is printed for each team, that is headlined by the team's name and sport-wide team number. A line is then printed for each game the team will play, giving the opponent, and the time, day of week, month, date, and court that it is scheduled on.

Finally, the routine asks whether the manager would like a listing of the full schedule printed. Generally, he will only request this after the last division has been scheduled, but the option is always open. The data providing the basis for this listing comes from the SCHED array, in contrast to the first two listings which are taken out of DIVSCHED. The format of the listing has the courts listed across the top and the days, and timeslots in each, listed chronologically down the left-hand side. If a game has been scheduled at a particular time on a given court, the teams playing in that game will be listed (as a pair of sport-wide team numbers) in that row and column. If no game is scheduled in that slot, the pair "0 - 0" will appear instead.

RESULTS

A number of goals were presented in the Problem Statement section of this thesis. With the possible exception of modifiability all of these goals were realized in the code produced. This code provides all of the flexibility needed to handle every sport and gives the manager all of the schedule-shaping power that was deemed desirable in the Problem Statement. In addition, the program keeps required input to a minimum and clearly prompts for each item so that it is easy for anyone to use this program. Substantial efforts were made to produce modifiable code -- in particular the code was broken down into 10 modules. However, it is always difficult to follow the functioning of code written by another person, and the length of this program exacerbates this problem. My feeling is that modifying this code would not be trivial, but has been easier through the efforts made to modularize it.

To demonstrate the extent to which various goals have been met, a set of test runs has been included in Appendix D. Important features of these runs are pointed out in the following paragraphs. The data provided for these test cases was designed to highlight various features of the program one-by-one. For this reason, most of the test cases fail to take advantage of all but one or two of the available constraints, providing results that don't correlate very well with actual scheduling tasks, but are much easier to follow.

The primary goal mentioned in the outlining of the task was to provide a high degree of "user friendliness". Several examples of how well this goal was realized can be found on the first page of output included in Appendix D. First, note the clarity of the prompt statements used, and particularly the use of sample answers to exemplify what sort of

input is being requested. Secondly, the minimal amount of input required by the program can be seen by observing how calendar and facility-availability information is entered. To specify the season's calendar the user is asked for the type of sport week, the number of months in the season, and the first day and last date in each month. When it is given the start and end dates for the season, it has all the information it needs to produce an internal calendar. The specification of the time-slot availabilities for each court can be found on the second page of output. To minimize input, the user is asked only to specify a sample week of availabilities that will then be replicated through the season. These availabilities are entered as bit-strings to provide full flexibility of input in a compact form. Since it is easy to wind up with a typo in the course of entering a series of bit-strings, the program checks at the end for the correctness of all the entries, allowing the user to re-enter as many as he wishes. Note how this facility was advantageous in the entering of this data on the second page of output. After this sample week has been verified and replicated, the program allows the user to specify any full days to be removed from the schedule and any days which will have a modified set of time-slot availabilities on any court. These tasks of calendar creation and facility availability specification could not be implemented in such a way to provide the same flexibility of data creation with less required input.

A whole set of goals was presented in the Problem Statement under the topic of Control Over Scheduling Desired. Among the capabilities deemed desirable (all of which were implemented) were the specification of mandatory and preferentially weighted courts, the specification of preferentially weighted days of the week and times for scheduling of a

particular league's games, and the specification of days or parts of days in the season on which each was unable to play due to external conflicts.

The first test run schedules a 3 month-long weekend season for a sport containing 2 leagues of 2 divisions each. There are four time slots, and to keep things simple each of the 3 courts is available during all 4 slots on both Saturdays and Sundays. The first A-league division is scheduled and printing is then done of the schedule for that division and the schedule for each team within the division. Following that, the second A-league division is scheduled, and the division and full-sport (currently just A-league as that's all that's been scheduled) schedules are printed. This full-sport schedule provides graphic output that clearly shows how the resulting scheduling was shaped by specified constraints.

By looking at this full schedule (for A-league) and observing the positioning of teams 1 through 6 (the first division scheduled) we can see the effects of specifying mandatory courts, and day of week and time-slot preferences. Court 1 was specified as mandatory for A-league and no games in either division 1 or 2 were scheduled on the other two courts. Sunday was designated as a preferable day, and time-slots 3 and 4 on Sundays were also designated as preferable. Both of these preferential constraints were given a weight of 5, implying that if the first open slot found fails to satisfy one or both of these constraints it will search 5 calendar days further in attempt to find a slot that does satisfy them. The result of this is that the 3 games in the first block for division 1 are scheduled on Sunday, June 13 in time-slots 3, 4, and 1 respectively. When each of these three games is scheduled, the pro-

gram looks only for openings in the Cage and in each case the first open slot found is Slot 1 on June 12. This slot satisfies neither preferential constraint, and since each has a weight of 5 the program is willing to search through June 17 for a better fit. The program next notes an opening in Slot 1 on June 13 satisfying the day preference but not the time preference. In scheduling the first two games, this slot is not chosen since later in the same day a preferred slot is available. When the third game is to be scheduled, no slot satisfying both constraints can be found in the June 12 to June 17 interval, so the best fit found is settled for. This happens to be Slot 1 on June 13 since it satisfies the preferred day constraint. This process then repeats through the season with a spacing of 10 days kept between the end of one block and start of the next, until all 5 blocks of games have been scheduled. Note also that the last block of games contains pairings of team 1 versus team 2 and team 3 versus team 4 in order to provide for seeding.

When the second A-league division (teams 7-10) is scheduled with the same constraints, much the same process occurs. In this case, however, most of the choice slots are already taken and the program has to settle for less highly preferred ones. Since each block contains 2 games (4 teams/2), and there is only one slot available in the 5 day span that satisfies either constraint (Slot 2 on June 13 in the first block), the first game is scheduled in that slot, and the second game is scheduled in the first open slot that was found (Slot 1 of june 12 for the first block).

This first test run then schedules 2 B-league divisions. No mandatory court is specified for B-league, but court 1 is still most preferred (with a weight of 5 compared to 1 each for courts 2 and 3). Sunday is

again preferred, but this time has a weight of only 2. Since this weight is lower than the weight for the court preference, given a choice between satisfying either the court preference or the day preference it will choose the former. The first division contains 7 teams (teams 11-17) each required to play at least 5 games. The fact that there are an odd number of teams in the division results in one team being left out in each block of games, and necessitates the adding of an additional block to the schedule (Thus, there are 6 blocks for a 5 game season.) The result is that 18 games are scheduled, with 6 of the teams playing 5 games, and 1 of them playing 6. Prior to scheduling the division, the user states that team 1 in the division (team number 11 overall) cannot play on either June 12 or June 13, and that the minimum spacing between blocks should be 5 days (this value for MIN_SPACE insures that a team will not play twice on the same weekend, but still allows consecutive games to be scheduled on a Sunday and the following Saturday).

The scheduling of division 1 of B-league now proceeds with the most important preference being placement of games on court 1 and a secondary preference being the scheduling of games on Sunday. Since every slot on court 1 on every other Sunday has been filled with A-league games, the day preference is hard to meet without sacrificing the preferred court. Since the court preference has been deemed more important, most of the games for this division are scheduled on Saturdays, with all of them taking place on court 1. What happens with the first block of games is particularly interesting. Two of the three games are scheduled on Saturday, June 12 since open slots are available then on court 1, and though they do not satisfy the day preference there are no slots within the 5-day search interval that satisfy both constraints. While there is still one

more open slot on Court 1 on June 12, the game pitting teams 1 and 6 is not placed there. This is due to the fact that Team 1 specified June 12 and June 13 as days it could not play. Therefore, the first open slot its game could be scheduled in was on Saturday, June 19 and since slots were open on the very next day which satisfied both preferential constraints, it was scheduled on that Sunday.

Division 2 of B-league contains 6 teams (numbers 18-23) each of whom plays 3 games. Once again, Court 1 is given a weight of 5 and Sunday is given a weight of 2 (since this is still B-league). Time-constraints specified for this division consist of the first team's (team number 18 overall) inability to play on either June 12 or June 13 during time-slots 1 through 3. To demonstrate the program's ability to recover from an unsuccessfully scheduled division, it was originally given a MIN_SPACE of 87 days between blocks -- a restriction it clearly could not meet. The program then allowed for printing of constraint data and relaxing of constraints (neither of which was desired in this case) and then tried again after being given a new MIN_SPACE. This time 4 was chosen and the scheduling was successful. As with the scheduling of B-league division 1, the court preference still takes priority over the day preference. However, by this point Court 1 was so full that it was frequently impossible to find an open slot for it within the confines of the 5 day search limit. For this reason, many of the games for this division were scheduled on Court 2 on Sundays. The second block was even able to be scheduled on a Sunday on Court 1, since that was one of the Sundays skipped by the A-league teams in their every-other-weekend scheduling. What happened with the first block of this division is also interesting. Game 1 (22 vs. 23) was scheduled in the last remaining slot on Court 1 on Saturday, June 12. At this point no openings remained in the Cage (Court 1)

on that first weekend. The second and third games in the block (19 vs. 20 and 18 vs. 21 respectively) therefore had to be scheduled on one of the other courts, and the only preference that could be satisfied for them was the day of week preference. Therefore, Game 2 was scheduled in the first open slot on Court 2 on Sunday, June 13. Normally, Game 3 would have been scheduled in the following slot on the same day. However, Team 18 was playing in Game 3 and was unable to play in slots 1 to 3 on that date. Game 3 was therefore pushed back to slot 4 on that day and court.

The functioning of all of the desired control features was seen in this first test run. The second test run is included not to demonstrate further features, but simply to make a little clearer the way in which multiple constraints interact.

In this second run, 2 leagues of 1 division each were scheduled in a one month long weekend sport. Each of the 4 courts was available during all 4 time-slots with the exception of Court 1. Court 1 was available during only the first three slots on Sundays, and only the first two slots on Sunday, December 7.

The first league (Z-league) specified that Court 1 was to be weighted by 5 with the other courts receiving weights of 1, and that time-slots 3 and 4 on Sundays would get a weight of 3. The single division in this league contained 8 teams playing 3 games each, and was scheduled using a MIN_SPACE of 5. These constraints then favored first the scheduling of games on Court 1 and secondarily the scheduling of games in one of the last two time slots on Sundays. The general behavior these preferences resulted in was that the program would attempt first to schedule the first two time slots in each block in the last two slots on Court 1 on a

Sunday. Since Court 1 is not available during slot 4 on Sundays, only slot 3 can thus be filled. Since the weight for the preferred times is less than that for the preferred court and not sufficiently high to cause the search to proceed to the next weekend, the remaining 3 games are scheduled in the first 3 slots on the previous Saturday. Note that Sunday as a whole is not preferentially weighted, only its last two time slots. In the case of the first block all four games are scheduled on Court 1 on Saturday, December 6 since the normally available preferential slot on the following Sunday is cancelled for that week.

The second league (Q- league) specified a weight of 3 for Court 1 and 0 for the other courts; a preferential weighting of 2 for Saturdays as a whole; and a weighting of 5 for slots 2 and 3 on Saturdays. There were again 8 teams to schedule with 3 games each and a MIN_SPACE of 5. In this case the driving force is geared toward using slots 2 and 3 on Saturdays, regardless of what court is available (though Court 1 is preferable). As a result, each block of games for this division was scheduled on Courts 2 and 3, during time-slots 2 and 3 on Saturdays.

The resulting season schedule is printed at the end of the run. The important thing to note is that both divisions were scheduled in a fashion that seemingly ignored one of the preferential constraints. The reason in both cases was that the ignored constraint could not be satisfied at the same time as another more important constraint, and the higher priority won out. In the case of the first division, the preference for late Sunday time-slots was overruled to allow for the placement of all games on Court 1. In division 2, the desire to play games on Court 1 was suppressed to allow the scheduling of games during slots 2 and 3 on Saturdays.

SUGGESTIONS FOR IMPROVEMENT

There are three areas in which I feel valuable improvements could be made. The first would be the addition of a capability to revise the original schedule to take care of games that have been rained out. The second improvement would be the use of a more sophisticated weighting function that would not overlook potentially good solutions. The third would be to have the program provide more help to the user in determining problems that cause a particular division to be unsuccessfully scheduled. Each of these will now be discussed more fully.

As I mentioned in my description of the algorithm, the inclusion of re-scheduling capabilities for rained-out games is not that crucial to the program's usefulness. It is, however, a feature that would be nice to have in some cases. When only a few games need to be rescheduled, the manager would probably find it much less bothersome to do this by hand than to go run the program. In cases where a large number of games were rained out, though, it might be easier to have the program do the work, and might also result in a better quality result through the satisfaction of preferential constraints.

A major improvement would be the replacement of the current weighting functions with bell-shaped functions. This would result in the search for a given game's placement not being strictly limited to the days following some arbitrary cutoff (i.e. currently blocks begin at MIN_SPACE days past the end of the last block). The apex of the function could be placed at the same position the start of the blocks currently are, providing the same optimal scheduling position in the absence of preferential constraints. By having this function rise sharply on the left and trail off more slowly on

the right, a nice spacing between blocks could still be kept without ruling out from consideration slots that occur just one or two days earlier than the ideal positioning. In other words, it is generally better to have a little extra spacing between games than a little less, but in some cases a preferential constraint might be satisfied one day earlier and not again until many days later.

The program could be much more helpful to the user in those cases where a division is not successfully scheduled. Cureently, the raw data that governs whether or not the scheduling will be successful is available for his perusal, but this is not going to be of much use without an understanding of the internal workings of the program. Instead, the current emphasis is on just reducing the minimum spacing requirement and trying again. This should solve the problem in most cases and has the advantage that the manager does not have to think too hard about where the problems might lay. However, if the program was capable of giving a clear analysis of just where the problem was and suggesting which constraint or constraints should be relaxed this would be clearly preferable.

REFERENCES

1. Richard Conway, David Gries, and David Wortman; Introduction to Programming Using PL/1 and SP/k; Winthrop Publishers. 1977

2. Steve Pettinato; "A Sports Scheduling System"; M.I.T. Bachelor's Thesis, Department of Electrical Engineering and Computer Science. 1980

APPENDIX A

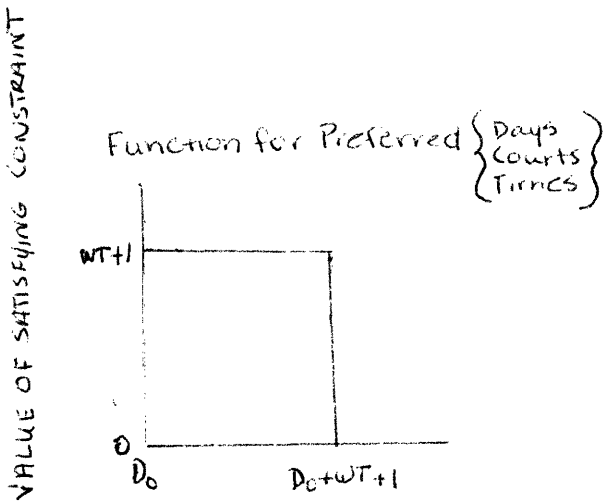
Weighting Functions and Evaluation of Optimal Time-Slot

Each of the preferential constraints used in this program is given a weight so that the importance of satisfying them can be determined. Since an effort is made to schedule each game as early as possible (as close to the start of the given block of games as possible), the degree of importance of satisfying a particular constraint can be naturally expressed as the number of days that the user is willing to postpone scheduling of a game in order to satisfy that constraint. Thus, if a constraint is given a weight of 10, the program will search through the days of the season until an open slot is found, and if the given slot does not satisfy that preferential constraint, the program will search up to 10 days farther in an attempt to find an open slot that does satisfy it.

An appropriate set of weighting functions must, therefore, result in the preferred slot being valued more highly up to the point where it is WEIGHT days past the non-preferred slot, and less highly after that. Such a set of functions is diagrammed on the following page. As one moves farther on in the season, the weight associated with the closeness to the start of the block decreases by one each day. It is set up, though, so that by the time the last day in the season is reached, this function still has a value of one. Thus, even if no preferential constraints are satisfied, there is still some reward given for finding any open slot that the game can be scheduled in.

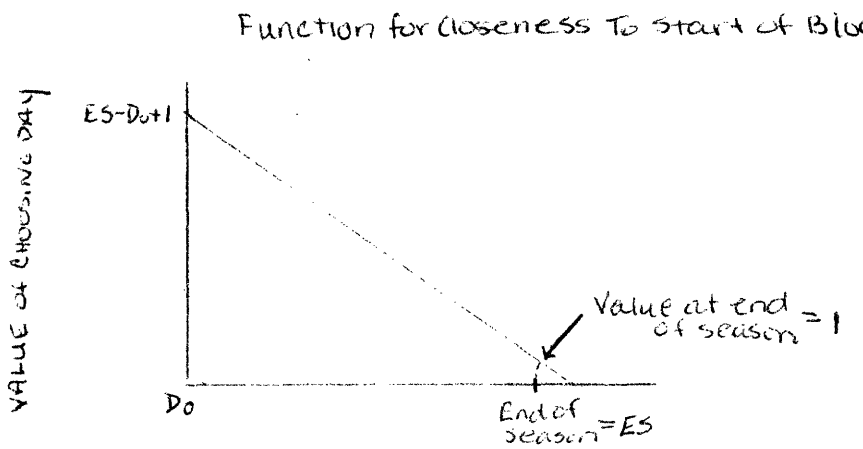
Each preferential constraint has a weighting function that is "square", having a constant value of WEIGHT + 1 over an interval of WEIGHT + 1 days.

WEIGHTING FUNCTIONS FOR PREFERENTIAL CONSTRAINTS

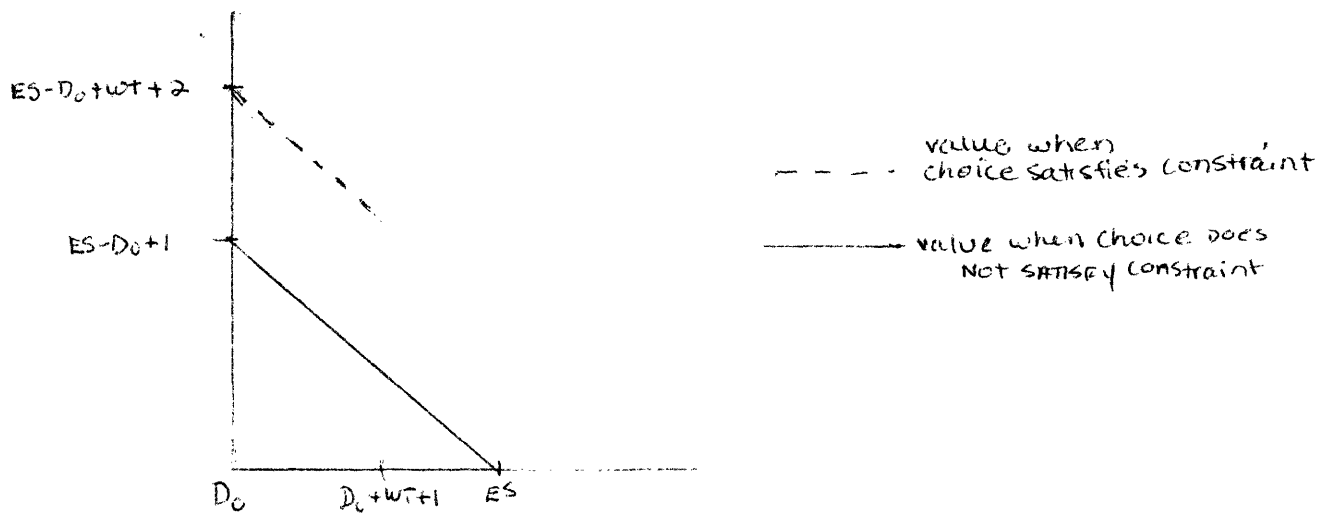


D_0 = Start of Block

WT = Weight assigned to Constraint



FULL EVALUATION FUNCTION FOR ONE PREFERENTIAL CONSTRAINT



ES = End of Season

Thus, any time a preferential constraint is satisfied, its value is added to the value of the closeness to the start of the block function. This second function is, therefore, skewed upward by a value of WEIGHT + 1, and since that function decreases by 1 with each day, the value skewed upward by satisfying the given constraint is thus highter than any normal (non-preference satisfying) value on the function as many as WEIGHT days prior.

The EVAL function used to determine the desirability of each slot examined simply adds the values of any satisfied preferential constraints to the closeness-to-start-of-block function. MAX_VAL adds the highest weights set for each of the different types of constraints that are applicable to the day passed to it. The resulting value is the value of the best slot on the best court on that day. No guarantee is made that this slot will be open -- MAX_VAL is simply determining the best value that could conceivably be achieved on that day without going through work of checking all of the court and team time availability data.

There are two considerations the program takes onto account when trying to determine how many days the search for a given slot should continue. The primary concern is simply that an available slot be found at some point that satisfies all of the mandatory constraints. This consideration is overriding until either a slot is found or the end of the season reached. The second consideration is really a combination of two things -- the desires to satisfy as many preferential constraints as possible and the desire to schedule the game in the earliest possible slot. This second consideration provides a limit on the length of the search done is situations not governed by the first concern. This limit is the quantity SEARCH_DAYS that is calculated at the end of the SCHEDULE routine and

placed in LEAGUE constraint structure. As explained above, in the case of a single preferential constraint, SEARCH_DAYS takes on a value one greater than the WEIGHT given to that constraint. When a set of preferential constraints are present, the number of days that will be searched is limited by the most heavily weighted one of the set (i.e. SEARCH_DAYS takes on the value $MAX(WT_1 + WT_2 + WT_3 + \ldots) + 1$).

APPENDIX B

PROGRAM LISTING

```
declare (misc_ptr, season_ptr) pointer;
declare 1 misc based(misc_ptr),
          2 sport_name char(16) varying,
          2 num_leagues fixed,
          2 league(5),
             3 name char(1),
             3 num_divs fixed,
          2 sport_wk,
             3 sw1 fixed,
             3 sw2 fixed,
             3 sw3 fixed,
          2 num_courts fixed,
          2 court(10),
             3 name char(8) varying,
             3 total_slots ,
                4 used fixed,
                4 left fixed,
          2 num_months fixed,
          2 first_date fixed,
          2 end_date fixed,
          2 month(5),
             3 name char(9) varying,
             3 first_day fixed,
             3 last_date fixed,
             3 seas,          /* d1 and d2 contain indices of */
                4 d1 fixed,     /* first and last days of month */
                4 d2 fixed,     /* (for use with SEASON array) */
          2 time_slots,
             3 length fixed decimal(3,1),
             3 num fixed,
             3 start fixed decimal(3,1);

declare 1 season based(season_ptr),
          2 num_days fixed,
          2 day(150),
             3 day_of_wk fixed,
             3 month fixed,
             3 date fixed,
             3 rel_date fixed,
             3 avail bit(1);
```

```
declare league_ptr pointer;
declare 1 league based(league_ptr),
          2 court,
             3 mandatory,
                4 num fixed,
                4 ct(10) fixed,
             3 pref,
                4 yes bit(1),
                4 ct(10) fixed,
                4 wt(10) fixed,
          2 day,
             3 num fixed,
             3 pref(5) fixed,
             3 wt(5) fixed,
          2 time_num fixed,
          2 time(5),
             3 day fixed,
             3 start_slt fixed,
             3 end_slt fixed,
             3 wt fixed,
          2 search_days fixed;
```

```
declare (div_ptr, div_s_ptr) pointer;

declare 1 division based(div_ptr),
          2 num_teams fixed,
          2 num_games fixed,
          2 team_name(8) char(10) varying,
          2 team_number(8) fixed;

declare 1 divsched based(div_s_ptr),
          2 n_games fixed,
          2 game(56),
             3 t11 fixed,
             3 t22 fixed,
             3 day fixed,
             3 ct fixed,
             3 time fixed;
```

```
declare sched_ptr pointer;

declare 1 sched(num_days,num_courts,time_slots.num) based(sched_ptr),
          2 t11 fixed,
          2 t22 fixed;
```

```
main: procedure options(main);

declare court_avail entry (pointer, pointer,
                           (150,10) bit(15) varying);
declare schedule entry (pointer,pointer,(150,10) bit(15) varying);
%include structures;
declare sysin file stream input;
declare sysprint file stream output;

declare facavail(150,10) bit(15) varying;
declare answer char(10) varying;
declare (i, j, k, ians, lg, div, max, teams_scheduled) fixed;
declare time fixed decimal (3,1);

declare struct_stor area automatic;
allocate misc in(struct_stor) set(misc_ptr);
allocate season in(struct_stor) set(season_ptr);

put skip list("Welcome to the intramural sports scheduling system.");
put skip edit("Used, properly this program should be of great assistanc"
                ,"e but you may")  (2 a);
put skip edit("run into problems if you have not first looked over the",
                " user's manual.")  (2 a);
put skip list("If you have not yet done so, you are advised to stop");
put skip edit("here and do some reading.  Do you wish to continue?",
                " (y/n) ")  (2 a);
get list(answer);
if answer = "n" then stop;
put skip list ("Please be careful to always enter data in the format");
put skip edit("specified by the question.  For instance, when responses"
                ," are given")  (2 a);
put skip list("in brackets at the end of the question, choose one of");
put skip list("them.  What is the sport name ?  ");
get list(misc.sport_name);
put skip list ("How many leagues (i.e. A,B,C, etc.) will there be ? ");
get list (misc.num_leagues);
put skip list ("You will now be asked for the name of each league and");
put skip list("the number of divisions in each.  Please input the ");
put skip list("leagues in the order you want them scheduled in. Also,");
put skip list("remember that divisions must have 4 to 8 teams in them");

do i=1 to misc.num_leagues;
   put skip list ("What is the name of league ", i," ?  ");
   get list (misc.league(i).name);
   put skip list ("How many divisions in league ", i, " ? ");
   get list (misc.league(i).num_divs);
   end;


put skip edit ("Is sport a weekend(1), weekday(2), or full-week(3)",
                          " sport? ")  (2 a);
get list (ians);
if ians=1
then do;
   misc.sport_wk.sw1 = 1;
   misc.sport_wk.sw2 = 7;
   misc.sport_wk.sw3 = 6;
   end;
else if ians=2
     then do;
```

```
            misc.sport_wk.sw1 = 2;
            misc.sport_wk.sw2 = 6;
            misc.sport_wk.sw3 = 1;
            end;
         else do;
            misc.sport_wk.sw1 = 1;
            misc.sport_wk.sw2 = 7;
            misc.sport_wk.sw3 = 1;
            end;

      put skip list ("How many months does the regular season span ? ");
      get list (misc.num_months);
      do i=1 to misc.num_months;
         put skip edit ("What is the name of month", i, " ? ") (a, f(2,0), a);
         get list (misc.month(i).name);
         put skip list ("What day of the week does this month start on? ");
         put skip list("Please give a number, i.e. Sunday=1, Friday=6, etc. ");
         get list (misc.month(i).first_day);
         put skip list ("How many days in this month ? i.e. 28,29,30,or 31  ");
         get list (misc.month(i).last_date);
         end;
      put skip edit("What date will season start on? (i.e. the first day",
                        " in ",month(1).name," that's in season) ") (4 a);
      get list (misc.first_date);
      put skip edit ("What date will season end on ? (no month,",
                                 " just date )")  (2 a);
      get list (misc.end_date);

      put skip list ("How many courts are available ? ");
      get list (misc.num_courts);
      put skip edit ("How long are the time-slots for each game?  Please resp"
                                 ,"ond with a fraction")  (2 a);
      put skip edit("representing the number of hours. (i.e. 90 minutes =",
                                 " 1.5  ")  (2 a);
      get list (misc.time_slots.length);
      put skip list ("When does the earliest time slot (on any day) start?");
      put skip edit (" ","[9:30 a.m. = 9.5,  1:00 p.m. = 13.0]  ")  (a(6), a);
      get list (misc.time_slots.start);
      put skip list ("When does the latest time-slot (on any day) end? ");
      get list (time);
      misc.time_slots.num = (time - misc.time_slots.start) /
                                 misc.time_slots.length ;

      /* given the above info. we can now determine what days will be in */
      /* the season and what courts will be available on each day.  This */
      /* info. will be stored in 'season' and 'facavail' respectively */


      call court_avail (misc_ptr, season_ptr, facavail);

      call schedule (misc_ptr, season_ptr, facavail);

      free misc;
      free season;

      end;      /* main */
```

```
court_avail: procedure (misc_ptr, season_ptr, facavail);

declare create_season entry (pointer, pointer);
%include structures;
declare sysin file stream input;
declare sysprint file stream output;

declare facavail (150, 10) bit(15) varying;
declare 1 court (10),
          2 day(7),
             3 day_avail bit(1),
             3 time_slot bit(15) varying;
declare (hour, imon, idate, iday, old_num_slots, new_num_slots) fixed;
declare (i, ict, last_slot, mins, d_o_w) fixed;
declare time fixed decimal(3,1);
declare (time_slot_string, new_slot_string, old_slot_string, nil_string)
                                        bit(15) varying;
declare ans char(10) varying;
declare min char(2);
declare day_name(7) char(9) varying initial("Sunday","Monday",
          "Tuesday","Wednesday","Thursday","Friday","Saturday");

do i = 1 to num_courts;
   court(i).total_slots.used = 0;
   court(i).total_slots.left = 0;
   end;

last_slot = misc.time_slots.num;
nil_string = ""b;
do i = 1 to last_slot;
   nil_string = nil_string || "0"b;
   end;

/* input names of facilities in default order of preference */

put skip(2) list ("Please enter the names of the courts in their");
put skip list ("default order of preference.");
do i = 1 to misc.num_courts;
   put skip list ("What is the name of facility ",i,"? (8 chars max)");
   get list (misc.court(i).name);
   end;

/* output time-slots so user will know index num. of each */

time = misc.time_slots.start;
do i = 1 to last_slot;
   hour = floor (time);
   mins = ( time - floor(time) ) * 60.0;
   if mins = 0 then min = "00";
               else min = "30";
   if hour > 12 then put skip edit("Time-slot ",i," is ",hour-12, ":",
                    min, " p.m.")  ( 2 (a, f(2,0)), 3 a);
                else if hour = 12
                    then put skip edit("Time-slot ",i," is ",hour,":",
                              min," p.m.")  (2 (a,f(2,0)), 3 a);
                    else put skip edit("Time-slot ",i," is ",hour, ":",
                              min, " a.m.")  ( 2 (a, f(2,0)), 3 a);
   time = time + misc.time_slots.length;
   end;
```

```
/* create sample-week of time-slot availabilities for each court */
put skip edit ("You will be asked to enter the availabilities of ",
                  "each of the courts")  (2 a);
put skip edit ("for the 'typical' days in the season.  Please enter",
                  " these availabilities as ")  (2 a);
put skip edit (last_slot, "-bit numbers with a 1 for each available",
                  "time-slot")  (f(2,0), 2 a);
put skip list ("and a 0 for each unavailable slot ");
do iday = misc.sport_wk.sw1 to misc.sport_wk.sw2 by misc.sport_wk.sw3;
   do ict = 1 to misc.num_courts;
      put skip edit ("Please enter the availability of court '",
               misc.court(ict).name, "' on a typical ", day_name(iday))
                  (a, a, a, a);
      get list (court(ict).day(iday).time_slot);
      end;           /* do ict */
   end;       /* do iday */

put skip list ("Were all strings entered correctly ? (y/n)");
get list (ans);
do while (ans = "n");
   put skip list ("Enter day of week to be changed as a number ");
   get list (iday);
   put skip list ("Enter court number to be changed ");
   get list (ict);
   put skip list ("What is the correct availability string ? ");
   get list (court(ict).day(iday).time_slot);
   put skip list ("Is everything correct now ? ");
   get list (ans);
   end;

/* determine available days in full season */

call create_season (misc_ptr, season_ptr);

/* apply sample week to season to get approximate time-slot */
/* availability for each court */

do iday = 1 to season.num_days;
   if season.day(iday).avail = "1"b
   then do;
      d_o_w = season.day(iday).day_of_wk;
      do ict = 1 to misc.num_courts;
         time_slot_string = court(ict).day(d_o_w).time_slot;
         facavail(iday, ict) = time_slot_string;
         do i = 1 to last_slot;
             if substr(time_slot_string, i, 1) = "1"b
             then court(ict).total_slots.left
                        = court(ict).total_slots.left + 1;
             end;    /* do i */
         end;        /* do ict */
      end;       /* then do (if season..) */
   else do ict = 1 to misc.num_courts;  /* day's availability is nil */
          facavail (iday, ict) = nil_string;
          end;       /* else do ict */
   end;        /* do iday */

/* now input exceptions to generalized court availabilities */

put skip(2) list ("Are there any days on which more or fewer time");
put skip list ("slots than usual will be open ? (y/n)");
```

```
get list (ans);

do while (ans = "y");
   put skip list ("Input the name of the court whose availability ");
   put skip list ("is to be changed.");
   get list (ans);
   do i = 1 to 10;
      if misc.court(i).name = ans then ict = i;
      end;
   put skip list ("Input the number of the month, and the date to be",
                  " modified.");
   get list (imon, idate);

   do iday = month(imon).seas.d1 to month(imon).seas.d2;
      if season.day(iday).date = idate
      then do;
          old_slot_string = facavail (iday, ict);
          old_num_slots = 0;
          do i = 1 to last_slot;
             if substr(old_slot_string, i, 1) = "1"b
                   then old_num_slots = old_num_slots + 1;
             end;   /* do i */
          put skip edit ("Please enter the availability of court ",
                misc.court(ict).name, " on ", month(imon).name, idate)
                  (a, a, a, a, f(3,0));
          put skip edit ("as a ", last_slot, "-bit number with a 1",
                "for each available time_slot and a 0") (a,f(2,0),2 a);
          put skip list ("for each unavailable slot");
          get list (new_slot_string);
          new_num_slots = 0;
          do i = 1 to last_slot;
             if substr (new_slot_string, i, 1) = "1"b
                   then new_num_slots = new_num_slots + 1;
             end;   /* do i */
          facavail (iday, ict) = new_slot_string;
          misc.court(ict).total_slots.left = court(ict).total_slots.left
                   + new_num_slots - old_num_slots;

          /* desired change made; exit day loop and check for */
          /* further changes */

          iday = 200;
          end;        /* if season.day(iday).... */
      end;            /* do iday */

   put skip list ("Are there other days that have fewer or more");
   put skip list ("time slots than normal ? (y/n)");
   get list (ans);
   end;               /* do while (ans = "y") */

return;
end;       /* court_avail */
```

```
create_season: procedure (misc_ptr, season_ptr);

%include structures;

declare sysin file stream input;
declare sysprint file stream output;
declare (sw1,sw2,sw3,date1,imon,iwk,idate,iday,incr) fixed;
declare (i,sundate,dow_last,rel_date,day_of_wk) fixed;
declare ans char(1);

season.num_days = 0;
imon, idate, iday = 1;
rel_date = 0;
date1 = misc.first_date;
day_of_wk = misc.month(1).first_day;
sw1 = misc.sport_wk.sw1;
sw2 = misc.sport_wk.sw2;
sw3 = misc.sport_wk.sw3;

/* Find date of Sunday starting first week of season */

do sundate = (2-day_of_wk) to 31 by 7;
   if (sundate+7) > misc.first_date then go to sundate_found;
   end;

sundate_found:

/* fill out (possibly partial) first week of month */

dow_last = 1;
idate = sundate;
misc.month(imon).seas.d1 = iday;
do day_of_wk = sw1 to sw2 by sw3;
   incr = day_of_wk - dow_last;
   rel_date = rel_date + incr;
   idate = idate + incr;
   if (idate > misc.month(imon).last_date) then go to end_month;
   if idate >= date1
   then do;
      season.day(iday).day_of_wk = day_of_wk;
      season.day(iday).month = imon;
      season.day(iday).date = idate;
      season.day(iday).avail = "1"b;
      season.day(iday).rel_date = rel_date;
      iday = iday + 1;
      end;          /* if idate */
   dow_last = day_of_wk;
   end;     /* do day_of_wk */
dow_last = dow_last - 7;

/* fill out remaining weeks of month */

do iwk = 2 to 5;
   do day_of_wk = sw1 to sw2 by sw3;
      incr = day_of_wk - dow_last;
      idate = idate + incr;
      rel_date = rel_date + incr;
      if (idate > misc.month(imon).last_date) then go to end_month;
      season.day(iday).day_of_wk = day_of_wk;
      season.day(iday).month = imon;
```

```
      season.day(iday).date = idate;
      season.day(iday).avail = "1"b;
      season.day(iday).rel_date = rel_date;
      iday = iday + 1;
      dow_last = day_of_wk;
      end;          /* do day_of_wk */
   dow_last = dow_last - 7;
   end;       /* do iwk */

end_month:

misc.month(imon).seas.d2 = iday -1;
imon = imon + 1;
if imon > misc.num_months then go to end_season;
date1 = 1;
/* find date of Sunday starting week containing first of next month */
sundate = 2 - (misc.month(imon).first_day);
rel_date = rel_date + (misc.month(imon-1).last_date+sundate) - idate;
go to sundate_found;

end_season:

/*  remove days that are past last_date in season */

do i = (iday-1) to (iday-33) by -1;
   if season.day(i).date <= misc.end_date then go to exceptions;
                                  else season.day(i).avail = "0"b;
   end;

exceptions:

season.num_days = i;
put skip(2) list ("You will now be asked for dates (e.g. Holidays)");
put skip list ("which fall on days normally part of the season, but");
put skip list ("on which no games will be played.  Note that this");
put skip list ("does not include days on which only some of the");
put skip list ("courts are unavailable.");
do imon = 1 to misc.num_months;
   put skip list ("Are there any unavailable days in ",
              misc.month(imon).name, " ?");
   get list (ans);
   do while (ans = "y");
      put skip list ("What is the date ?");
      get list (idate);
      do iday = month(imon).seas.d1 to month(imon).seas.d2;
          if idate = season.day(iday).date
          then season.day(iday).avail = "0"b;
          end;
      put skip list ("Any more unavailable days in ",
                     misc.month(imon).name, " ? (y/n)");
      get list (ans);
      end;          /* do while */
   end;             /* do imon */

end;      /* create_season */
```

```
schedule: procedure( misc_ptr, season_ptr, facavail);

declare div_sched entry (pointer,pointer,pointer,pointer,fixed,fixed,
                         fixed, (150,10) bit(15) varying);

%include structures;
%include league_dcl;
%include sched_dcl;
declare sysin file stream input;
declare sysprint file stream output;

declare facavail(150,10) bit(15) varying;
declare answer char(10) varying;
declare (i, j, k, lg, div, max, teams_scheduled) fixed;
declare storagge area automatic;
allocate league in(storagge) set(league_ptr);
allocate sched in(storagge) set(sched_ptr);

do i = 1 to season.num_days;
   do j = 1 to misc.num_courts;
      do k = 1 to misc.time_slots.num;
          sched (i,j,k).t11 = 0;
          sched (i,j,k).t22 = 0;
          end;
      end;
   end;
teams_scheduled = 0;

do lg=1 to misc.num_leagues;
   put skip(2) edit ("We are now ready to schedule ",
                     misc.league(lg).name, "-league")  (3 a);
   put skip edit ("Is there a mandatory court (or courts) for this",
                         " league? (y/n) ")  (2 a);
   get list (answer);
   if answer = "y"
   then do;
      put skip list ("How many ? ");
      get list (league.court.mandatory.num);
      put skip edit ("Please input the numbers of these courts in",
                         " preferential order")  (2 a);
      do i=1 to league.court.mandatory.num;
         put skip edit("Court ", i, " ? ")  (a, f(2,0), a);
         get list (league.court.mandatory.ct(i));
         end;
      end;        /* then do */
   else do;
      league.court.mandatory.num = 0;
      put skip edit ("Are there preferred courts for this league?  (If",
                         " not, the order in which the")  (2 a);
      put skip edit("court names were originally entered will be used",
                         " as the default.)  ")  (2 a);
      get list (answer);
      if answer = "y"
      then do;
         put skip edit ("Enter the court numbers in order of desira",
                         "bility.  Along with each")  (2 a);
         put skip edit("court number please enter a weighting factor",
                         " reflecting how desirable")  (2 a);
         put skip edit("that court is.  The weight represents the",
                         " number of days that")  (2 a);
```

```
      put skip edit("you are willing to push a game back in the",
                        " schedule in order to have")  (2 a);
      put skip list("it scheduled on that court.");
      do i=1 to misc.num_courts;
         put skip edit ("Court ", i, " ? ")  (a, f(2,0), a);
         get list (league.court.pref.ct(i));
         put skip list ("What is the weight for this court ? ");
         get list (league.court.pref.wt(i));
         end;              /* do i */
      league.court.pref.yes = "1"b;
      end;        /* then do */
   else league.court.pref.yes = "0"b;
   end;           /* else do (if mand.courts...) */

put skip edit ("Are there any preferred days of the week for schedul"
      ,"ing",misc.league(lg).name,"-league games ? (y/n) ") (4 a);
get list (answer);
if answer = "y"
then do;
   put skip list ("How many preferred days are there ? ");
   get list (league.day.num);
   put skip edit ("Please enter these preferred days in order of",
                        " desirability.  Along with")  (2 a);
   put skip edit("each day enter a weighting factor corresponding to"
                        ," the number of days you")  (2 a);
   put skip edit("would be willing to delay the scheduling of a",
                        " particular game to make")  (2 a);
   put skip list("it fall on that day. ");
   do i=1 to league.day.num;
      put skip edit ("Day ", i, " ? ")  (a, f(2,0), a);
      get list (league.day.pref(i));
      put skip list ("Weighting factor ? ");
      get list (league.day.wt(i));
      end;        /* do i */
   end;           /* then do */
else league.day.num = 0;

put skip edit ("Are there any times that are preferred for schedul",
          "ing", misc.league(lg).name, "-league games ? ")  (4 a);
get list (answer);
if answer = "y"
then do;
   put skip edit("Preferred times consist of a block of time-slots",
                        " on a particular day of")  (2 a);
   put skip edit("the week.  You may specify up to 5 such blocks by",
                        " giving the day, starting")  (2 a);
   put skip edit("slot and ending slot of each prefered block.  You",
                        " will also be asked to give")  (2 a);
   put skip edit("a weight to eacn block corresponding to the num",
                        "ber of days a game's")  (2 a);
   put skip edit("scheduling may be delayed in order to fall into",
                        " that block.")  (2 a);
   put skip list ("How many such blocks do you wish to specify ? ");
   get list (league.time_num);
   do i=1 to league.time_num;
       put skip edit ("Enter the number corresponding to the day on",
                        " which")  (2 a);
       put skip edit("block ", i, "(in preferential order) occurs. ")
                                          (a, f(2,0), a);
       get list (league.time(i).day);
```

```
            put skip edit ("Enter the number of the time-slot which ",
                                "begins this block." )  (2 a);
            get list (league.time(i).start_slt);
            put skip edit ("Enter the number of the last time-slot in",
                                " block." )  (2 a);
            get list (league.time(i).end_slt);
            put skip list ("Weight for this block ? ");
            get list (league.time(i).wt);
            end;        /* do i */
         end;           /* then do */
      else league.time_num = 0;

      /* Use weights entered above to determine the number of schedule */
      /* days that will be searched in looking for each game"s most */
      /* appropriate time-slot (best-fit)   */

      max = 0;
      if league.court.pref.yes = "1"b
      then do i=1 to misc.num_courts;
            if league.court.pref.wt(i) > max
                  then max = league.court.pref.wt(i);
            end;
      if league.day.num > 0
      then do i=1 to league.day.num;
            if league.day.wt(i) > max  then max = league.day.wt(i);
            end;
      if league.time_num > 0
      then do i=1 to league.time_num;
            if league.time(i).wt > max  then max = league.time(i).wt;
            end;
      league.search_days = max + 1;

      /* now that parameters for league are specified, loop through all */
      /* the divisions in this league and schedule each one. */

      do div=1 to misc.league(lg).num_divs;
         call div_sched(misc_ptr, season_ptr, league_ptr, sched_ptr, lg,
                                div, teams_scheduled, facavail);
         end;           /* do div */

      end;    /* do lg */

   free league;
   free sched;

   end;       /* schedule */
```

```
div_sched: procedure (misc_ptr, season_ptr, league_ptr, sched_ptr,
                                  lg, div, teams_scheduled, facavail);

declare team_pair entry (fixed, fixed, (14, 4, 2) fixed );
declare eval entry (fixed, fixed, fixed, fixed, pointer, pointer, fixed)
             returns (fixed);
declare max_val entry (fixed, fixed, pointer, pointer) returns(fixed);
declare relax entry (pointer, pointer);
declare printout entry (pointer,pointer,pointer,pointer,pointer,
                                          fixed, fixed);

%include league_dcl;
%include structures;
%include div_dcl;
%include sched_dcl;
declare sysin file stream input;
declare sysprint file stream output;

declare 1 team(8),
          2 date(season.num_days),
             3 play_day bit(1),
             3 play_time bit(15) varying;

declare 1 change(56),
          2 day fixed,
          2 ct fixed,
          2 time fixed;

declare 1 start_week,
          2 mon fixed,
          2 rel_date fixed;

declare stowe area automatic;
allocate division in(stowe) set(div_ptr);
allocate divsched in(stowe) set(div_s_ptr);

declare num_changes fixed initial(1);
declare (lg, div, slt1, slt2, num_weeks, ict, value, best_fit) fixed;
declare (best_fit_day, best_fit_time, best_fit_ct, i, t, t1, t2) fixed;
declare (ts1, ts2, current_day, d, imon, idate, week, gm,num_cts) fixed;
declare (games_per_week, len, ts, start_day,teams_scheduled) fixed;
declare (j,k,iday,day1,min_space, end_last_week, end_this_week) fixed;
declare (court_list(10), divgame, high) fixed;
declare pairings(14,4,2) fixed;
declare (facavail(150,10), one_string, nil_string) bit(15) varying;
declare availability bit(15);
declare ans char(1);

nil_string, one_string = ""b;
do i = 1 to misc.time_slots.num;
   nil_string = nil_string || "0"b;
   one_string = one_string || "1"b;
   end;
divsched.n_games = 0;
do i = 1 to 56;
   divsched.game(i).t11 = 0;
   divsched.game(i).t22 = 0;
   end;

put skip(2) list("We are now ready to schedule division number ",div);
```

```
put skip edit ("        of ", misc.league(lg).name, "-league ") (3 a);
put skip list ("How many teams in this division? (must be 4 to 8) ");
get list (division.num_teams);
put skip list("How many games will each team play? ");
get list (division.num_games);
do i=1 to division.num_teams;
   put skip edit ("What is the name of team ", i, " ? ") (a, f(1,0), a);
   get list (division.team_name(i));
   end;

divsched.n_games = (num_games * num_teams) / 2;
if num_teams = 5 then divsched.n_games = (num_games + 1) * 2;
if num_teams = 7 then divsched.n_games = (num_games + 1) * 3;

/* set up team_pairings array for this division, given number of */
/* teams and number of games each will play */

call team_pair (division.num_teams, division.num_games, pairings);

/* input time constraints for each team in division */

do t=1 to num_teams;
   division.team_number(t) = t + teams_scheduled;
   do d=1 to num_days;
      team(t).date(d).play_day = "1"b;
      team(t).date(d).play_time = one_string;
      end;
   end;
put skip list ("Are there any days in the season on which one of");
put skip list ("the teams in this division cannot play? (y/n) ");
get list (ans);
do while (ans = "y");
   put skip list ("What team? (enter its number) ");
   get list (t);
   put skip list ("What month and date is ", team_name(t) );
   put skip list ("       not able to play on? (enter as numbers) ");
   get list (imon, idate);
   do i = month(imon).seas.d1 to month(imon).seas.d2;
      if season.day(i).date = idate
      then do;
          iday = i;
          i = 150;       /* exit loop */
          end;
      end;
   team(t).date(iday).play_day = "0"b;
   team(t).date(iday).play_time = nil_string;
   put skip list ("Are there more days on which teams in this");
   put skip list ("          division cannot play? (y/n) ");
   get list (ans);
   end;              /* do while */

put skip(2) list ("Are there any time_slots (or blocks of slots)");
put skip list ("     on which teams in this division cannot play? ");
get list (ans);
do while (ans = "y");
   put skip list ("What team? (enter number) ");
   get list (t);
   put skip list ("What month and date? (enter month as number) ");
   get list (imon, idate);
   do i = misc.month(imon).seas.d1 to month(imon).seas.d2;
```

```
      if season.day(i).date = idate
      then do;
          iday = i;
          i = 150;  /* exit loop */
          end;      /* then do */
      end;          /* do i */
   put skip list ("Now enter the numbers corresponding to the first");
   put skip list ("    and last time-slots in the unavailable block ");
   get list (slt1, slt2);
   do i = slt1 to slt2;
      substr(team(t).date(iday).play_time, i, 1) = "0"b;
      end;
   put skip list("Are there more blocks of time slots on which teams");
   put skip list ("       in this division cannot play? (y/n) ");
   get list (ans);
   end;   /* do while */

/* Create court_list for division.  Will include mandatory courts if */
/* such exist, and preferential ordering of all courts otherwise */

if league.court.mandatory.num > 0
then do;
   num_cts = league.court.mandatory.num;
   do i = 1 to num_cts;
      court_list(i) = league.court.mandatory.ct(i);
      end;
   end;
else do;
   num_cts = misc.num_courts;
   if league.court.pref.yes = "1"b
   then do i = 1 to num_cts;
      court_list(i) = league.court.pref.ct(i);
      end;
   else do i = 1 to num_cts;
      court_list(i) = i;
      end;
   end;      /* else do */

schedule:

/* schedule season for this division one 'week' at a time */

put skip list ("How many days would you like to leave between");
put skip list ("    consecutive games for each team in this division ");
get list (min_space);
if (division.num_teams = 5) | (division.num_teams = 7)
then do;
   num_weeks = division.num_games + 1;
   if division.num_games >= division.num_teams
                              then num_weeks = num_weeks + 1;
   games_per_week = (division.num_teams - 1) / 2;
   end;
else do;
   num_weeks = division.num_games;
   games_per_week = division.num_teams / 2;
   end;

end_last_week = season.day(1).rel_date - min_space;
start_week.mon = 1;
```

```
do week = 1 to num_weeks;
   end_this_week, start_week.rel_date = end_last_week + min_space;
   do i = misc.month(start_week.mon).seas.d1 to 150;
      if season.day(i).rel_date >= start_week.rel_date
      then do;
          start_day = i;
          if i > misc.month(start_week.mon).seas.d2 then
                          start_week.mon = start_week.mon + 1;
          i = 150;
          end;
      end;          /* do i */

   /* loop through games to be scheduled in this week (block) */

   do gm = 1 to games_per_week;
      t1 = pairings (week, gm, 1);
      t2 = pairings (week, gm, 2);
      current_day = start_day;
      /*  DAY1 records first slot found for scheduling current game. */
      /*  Until one is found, it is set at end_of_season since search */
      /*  is cut off at (DAY1 + SEARCH_DAYS */
      day1 = season.num_days;
      best_fit = 0;

new_day:

   /* If we have reached end_of_season or have gone SEARCH_DAYS past */
   /* the first open slot found, we should schedule game in the best */
   /* fitting position found (unless BEST_FIT = 0, ==> end_of_season */
   /* with no slot found. In that case, scheduling bombed.) */

   if (current_day > season.num_days) |
      (day(current_day).rel_date - day(day1).rel_date > search_days)
   then do;
      if best_fit = 0 then go to bomb;
      else do;
                  /*******   Record game in best slot *******/

        substr(facavail(best_fit_day,best_fit_ct),best_fit_time,1)="0"b;
        sched (best_fit_day, best_fit_ct, best_fit_time).t11 =
                                        division.team_number(t1);
        sched (best_fit_day, best_fit_ct, best_fit_time).t22 =
                                        division.team_number(t2);
        divgame = (week-1)*games_per_week + gm;
        divsched.game(divgame).t11 = t1;
        divsched.game(divgame).t22 = t2;
        divsched.game(divgame).day = best_fit_day;
        divsched.game(divgame).ct = best_fit_ct;
        divsched.game(divgame).time = best_fit_time;

        /* must keep track of changes made to court availabilities in */
        /* case division must be rescheduled (so they can be undone) */
          change(num_changes).day = best_fit_day;
          change(num_changes).ct = best_fit_ct;
          change(num_changes).time = best_fit_time;
          num_changes = num_changes + 1;
          court(best_fit_ct).total_slots.left =
                          court(best_fit_ct).total_slots.left - 1;
          court(best_fit_ct).total_slots.used =
                          court(best_fit_ct).total_slots.used + 1;
```

```
      if season.day(best_fit_day).rel_date > end_this_week
            then end_this_week = season.day(best_fit_day).rel_date;
      go to new_game;
                              /***** Finished Recording  *******/

      end;          /* else do (if best_fit = 0) */
   end;             /* then do (if day(current_day).rel_date)... */

if (team(t1).date(current_day).play_day = "0"b) |
   (team(t2).date(current_day).play_day = "0"b)
then do;
   current_day = current_day + 1;
   go to new_day;
   end;

/* check to see if game scheduled on this day could possibly */
/* improve on current best_fit */

high = max_val(current_day, day1, season_ptr, league_ptr);
if best_fit > high
then do;
   current_day = current_day + 1;
   go to new_day;
   end;

/* Examine court and time-slot combnations for this day */

do ict = 1 to num_cts;
   /* look for acceptable time-slot on this day and court. */
   /* look first for preferred slots */
   availability = team(t1).date(current_day).play_time &
                  team(t2).date(current_day).play_time &
                  facavail(current_day, court_list(ict));
   if league.time_num > 0
   then do i = 1 to league.time_num;
       if season.day(current_day).day_of_wk = league.time(i).day
       then do;
          ts1 = league.time(i).start_slt;
          ts2 = league.time(i).end_slt;
          do ts = ts1 to ts2;
             if substr(availability, ts, 1) = "1"b
             then do;
                value = eval (ts, current_day, day1,court_list(ict),
                          season_ptr,league_ptr,misc.num_courts);
                if value > best_fit
                then do;
                   /* if this is first open slot found for this */
                   /* game, record day in DAY1 */
                   if best_fit = 0 then day1 = current_day;
                   best_fit = value;
                   best_fit_day = current_day;
                   best_fit_time = ts;
                   best_fit_ct = ict;
                   end;
                go to day_done;
                end;       /* then do (if substr) */
             end;          /* do ts */
          end;       /* then do (if season...) */
       end;       /* do i */
```

```
            /* either no success with preferred slots or no */
            /* preferred slots exist - loop all slots */

         do ts = 1 to misc.time_slots.num;
            if substr(availability, ts, 1) = "1"b
            then do;
               value = eval(ts,current_day,day1,court_list(ict),
                            season_ptr, league_ptr, misc.num_courts);
               if value > best_fit
               then do;
                  /* if this is first open slot found for this */
                  /* game, record day in DAY1 */
                  if best_fit = 0 then day1 = current_day;
                  best_fit = value;
                  best_fit_day = current_day;
                  best_fit_time = ts;
                  best_fit_ct = ict;
                  end;
               if value = high     /* max_val for day has been obtained */
                    then go to day_done;
                    else go to new_court;     /* look for pref. time */
               end;   /* then do (if substr) */
            end;      /* do ts */

new_court:

      /* try a new court on this day.  Look first for a prefered */
      /* time-slot on that day -- if that fails, look for any slot */
      /* open on that court -- if that fails, go to new court again. */

      end;           /* do ict */

day_done:

   /* have gone through all appropriate courts and time-slots */
   /* on this day; try new day */
   current_day = current_day + 1;
   go to new_day;

new_game:
/* jump to here after a game has been scheduled */
   end;              /* do gm */

   end_last_week = end_this_week;
   end;              /*do week */

/* all scheduling for this division has been completed */


call printout (misc_ptr, season_ptr, div_ptr, div_s_ptr, sched_ptr,
                                                        lg, div);

teams_scheduled = teams_scheduled + division.num_teams;

free division;
free divsched;

return;
```

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

bomb:
/* if control passes here, division scheduling could not be completed */
/* try new smaller min-space between blocks (weeks) of games */

put skip list (" Scheduling of division was unsuccessful with a");
put skip edit ("min-space of ", min_space, "Would you like more info? ")
                                          (a, f(2,0), a);
get list (ans);
if ans = "y"
then do;
   put skip data (week, start_week, end_last_week, end_this_week);
   put skip list ("More info ? (facility availabilities) ");
   get list (ans);
   if ans = "y"
   then do;
      put skip data (start_day, current_day, court(*).total_slots);
      do i = start_day to current_day;
          put skip data (facavail (i,*));
          end;
      end;           /* then do */
   end;     /* then do */
put skip list ("Would you like to rerun with a new min-space? (y/n) ");
get list (ans);
if ans = "y"
then do;           /* re-open slots that were used during bomb run */
   call relax (misc_ptr, league_ptr);
   do i = 1 to (num_changes - 1);
      substr(facavail(change(i).day, change(i).ct), change(i).time, 1)
                                                        = "1"b;
      sched(change(i).day,change(i).ct,change(i).time).t11 = 0;
      sched(change(i).day,change(i).ct,change(i).time).t22 = 0;
      divsched.game(i).t11 = 0;
      divsched.game(i).t22 = 0;
      court(change(i).ct).total_slots.left =
                  court(change(i).ct).total_slots.left + 1;
      court(change(i).ct).total_slots.used =
                  court(change(i).ct).total_slots.used - 1;
      end;           /* do i */
   num_changes = 1;
   go to schedule;
   end;     /* then do */

end;       /* div_sched */
```

```
team_pair: procedure (num_teams, num_games, pairings);

declare sysprint file stream output;

declare (num_teams, num_games, games_per_week, i, week, game) fixed;
declare (array_lines, first_line, n_games) fixed;
declare pairings (14,4,2) fixed;
declare index (14) fixed;
declare array (7,4,2) fixed;
declare t(5) label;

if num_teams < 4 | num_teams > 8 then go to bad;
          else go to t(num_teams - 3);

t(1): call four(array_lines, games_per_week, array, i);
      go to good;
t(2): call five(array_lines, games_per_week, array, i);
      go to good;
t(3): call six(array_lines, games_per_week, array, i);
      go to good;
t(4): call seven(array_lines, games_per_week, array, i);
      go to good;
t(5): call eight(array_lines, games_per_week, array, i);
      go to good;

bad: put skip list ("Too many or too few teams");
     return;

good:

/* determine first-line of (raw data) 'array' to use such that */
/* last line will be placed in last week in 'pairings'.  This is */
/* necessary to utilize seeding capability of scheduling */

n_games = num_games + i;
first_line = mod (array_lines-mod(n_games,array_lines),array_lines);
if first_line = 0 then first_line = array_lines;

/* fill 'index' array with the line numbers of 'array' to be */
/* used in filling 'pairings' array */

do i = 1 to n_games;
   index(i) = mod (first_line + i, array_lines);
   if index(i) = 0 then index(i) = array_lines;
   end;

/* now fill 'pairings' */

do week = 1 to n_games;
   do game = 1 to games_per_week;
      pairings(week, game, 1) = array(index(week), game, 1);
      pairings(week, game, 2) = array(index(week), game, 2);
      end;
   end;

return;

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

four: procedure(array_lines, games_per_week, array, i);
```

```
declare (array_lines, games_per_week, i) fixed;
declare array (7,4,2) fixed;
declare a_ray(7,4,2) fixed   initial (2,3,1,4,0,0,0,0,
          2,4,1,3,0,0,0,0,   3,4,1,2,0,0,0,0,   0,0,0,0,0,0,0,0,
          0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0);
array = a_ray;
array_lines = 3;
games_per_week = 2;
i = 0;

return;
end;

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

five: procedure(array_lines, games_per_week, array, i);

declare (array_lines, games_per_week, i) fixed;
declare array (7,4,2) fixed;
declare a_ray (7,4,2) fixed   initial (2,4,1,5,0,0,0,0,
          2,3,1,4,0,0,0,0,   4,5,1,3,0,0,0,0,   3,5,1,2,0,0,0,0,
          2,4,2,5,0,0,0,0,   0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0);
array = a_ray;
array_lines = 5;
games_per_week = 2;
i = 1;

return;
end;

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

six: procedure(array_lines, games_per_week, array, i);

declare (array_lines, games_per_week, i) fixed;
declare array (7,4,2) fixed;
declare a_ray (7,4,2) fixed   initial (3,4,2,5,1,6,0,0,
          3,6,2,4,1,5,0,0,   5,6,2,3,1,4,0,0,   4,5,2,6,1,3,0,0,
          3,5,4,6,1,2,0,0,   0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0);
array = a_ray;
array_lines = 5;
games_per_week = 3;
i = 0;

return;
end;

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

seven: procedure(array_lines, games_per_week, array, i);

declare (array_lines, games_per_week, i) fixed;
declare array (7,4,2) fixed;
declare a_ray (7,4,2) fixed   initial (3,5,2,6,1,7,0,0,
          3,4,2,5,1,6,0,0,   6,7,2,4,1,5,0,0,   5,7,2,3,1,4,0,0,
          4,7,5,6,1,3,0,0,   4,6,3,7,1,2,0,0,   4,5,3,6,2,7,0,0);
array = a_ray;
array_lines = 7;
games_per_week = 3;
```

```
        i = 1;

        return;
        end;

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

        eight: procedure(array_lines, games_per_week, array, i);

        declare (array_lines, games_per_week, i) fixed;
        declare array (7,4,2) fixed;
        declare a_ray (7,4,2) fixed   initial (4,5,3,6,2,7,1,8,
                  4,8,3,5,2,6,1,7,   4,7,3,8,2,5,1,6,   6,8,3,7,2,4,1,5,
                  5,8,6,7,2,3,1,4,   4,6,5,7,2,8,1,3,   5,6,3,4,7,8,1,2);
        array = a_ray;
        array_lines = 7;
        games_per_week = 4;
        i = 0;

        return;
        end;

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

        end;       /* team_pair */
```

```
max_val: procedure (iday, day1, season_ptr, league_ptr)
          returns (fixed);

%include structures;
%include league_dcl;

declare (iday, rel_date1, best_value, i, days_from_start) fixed;
declare (max_weight, rel_date_now, day1, end_season) fixed;

/* Determine closeness of current day to start of block.  Value */
/* decreases linearly with distance, reaching 0 at end of season. */
/* This rewards any possible scheduling and favors tight packing. */

rel_date1 = season.day(day1).rel_date;
rel_date_now = season.day(iday).rel_date;
days_from_start = rel_date_now - rel_date1 - 1;
end_season = season.day(season.num_days).rel_date;
best_value = end_season - season.day(iday).rel_date + 1;

/* determine maximum possible value of satisfying court preference */

if (league.court.mandatory.num = 0) & (league.court.pref.yes = "1"b)
then do;
   max_weight = league.court.pref.wt(1);
   if max_weight > days_from_start
          then best_value = best_value + max_weight + 1;
   end;

/* determine max. possible value of satisfying day preference */

do i = 1 to league.day.num;
   if league.day.pref(i) = season.day(iday).day_of_wk
   then do;
      max_weight = league.day.wt(i);
      if max_weight > days_from_start
          then best_value = best_value + max_weight + 1;
      i = 6;          /* exit from loop */
      end;
   end;

/* determine max. possible value of satisfying time_slot preference */

do i = 1 to league.time_num;
   if league.time(i).day = season.day(iday).day_of_wk
   then do;
      max_weight = league.time(i).wt;
      if max_weight > days_from_start
          then best_value = best_value + max_weight + 1;
      i = 6;          /* exit from loop */
      end;
   end;

return (best_value);

end;       /* max_val */
```

```
eval: procedure (ts, iday, day1, ict, season_ptr, league_ptr, num_cts)
                    returns (fixed);

%include structures;
%include league_dcl;

declare (ts, iday, day1, ict, i, days_from_start, rel_date1) fixed;
declare (rel_date_now, end_season, value, weight, num_cts) fixed;

/* Determine closeness of current_day to start of block.  Value */
/* decreases linearly with distance, reaching 0 at end of season. */
/* This rewards any possible scheduling and favors tight packing. */

rel_date1 = season.day(day1).rel_date;
rel_date_now = season.day(iday).rel_date;
days_from_start = rel_date_now - rel_date1 - 1;
end_season = season.day(season.num_days).rel_date;
value = end_season - season.day(iday).rel_date + 1;

/* if this is a preferred day add in corresponding weight */

do i = 1 to league.day.num;
   if league.day.pref(i) = season.day(iday).day_of_wk
   then do;
      weight = league.day.wt(i);
      if weight > days_from_start
          then value = value + weight + 1;
      i = 900;
      end;
   end;

/* if this is preferred court add in the corresponding weight */

if league.court.pref.yes = "1"b
then do i = 1 to num_cts;
   if league.court.pref.ct(i) = ict
   then do;
      weight = league.court.pref.wt(i);
      if weight > days_from_start
          then value = value + weight + 1;
      i = 900;
      end;
   end;

/* if this is preferred time-slot, add in corresponding weight */

do i = 1 to league.time_num;
   if (league.time(i).day = season.day(iday).day_of_wk)
      & (league.time(i).start_slt <= ts)
      & (league.time(i).end_slt >= ts)
   then do;
      weight = league.time(i).wt;
      if weight > days_from_start
          then value = value + weight + 1;
      i = 900;
      end;
   end;

return (value);

end;
```

```
relax: procedure (misc_ptr, league_ptr, court_list);

%include structures;
%include league_dcl;
declare sysin file stream input;
declare sysprint file stream output;

declare ans char(1);
declare (i, ict, court_list(10)) fixed;

if court.mandatory.num > 0
then do;
   put skip edit ("There are currently ", court.mandatory.num,
             " mandatory courts.  They are:")  (a, f(1,0), a);
   put skip edit (" ", "COURT", "SLOTS USED", "SLOTS LEFT")
                           (a(5), a(8), 2 a(13));
   do i = 1 to mandatory.num;
      ict = mandatory.ct(i);
      put skip edit (" ", court(ict).name, court(ict).total_slots.used,
                court(ict).total_slots.left)  (a(4), a(10), 2 f(13,0));
      end;

   put skip edit ("Do you wish to get rid of the mandatory court req",
                     "uirements for this league ? (y/n)")  (2 a);
   get list (ans);
   if ans = "y"
   then do;
      court.mandatory.num = 0;
      put skip list ("Do you want to specify preferred courts ? ");
      get list (ans);
      if ans = "y"
      then do;
          put skip edit ("Enter the court numbers in order of desirabi",
                              "lity.  Also please give each a")  (2 a);
          put skip edit ("weight corresponding to the number of days",
                              " you would be willing to")  (2 a);
          put skip edit ("postpone the scheduling of a game to have it",
                              " put on this court." )  (2 a);
          do i = 1 to num_courts;
             put skip list ("court ", i, "? ");
             get list (league.court.pref.ct(i));
             court_list(i) = league.court.pref.ct(i);
             put skip list ("weight for this court ? ");
             get list (league.court.pref.wt(i));
             end;    /* do i */
          end;       /* then do */
      else do;
          do i = 1 to num_courts;
             court_list(i) = i;
             end;
          end;       /* else do */
   end;    /* then do (get rid of mand. courts) */

   else do;           /* no desire to get rid of mandatory courts */
      put skip edit ("Would you like to add to the list of mandatory ",
                              "courts ? (y/n) ")  (2 a);
      get list (ans);
      do while (ans = "y");
          put skip edit ("There are now ", court.mandatory.num,
                     " mandatory courts.  They are:")  (a, f(2,0), a);
```

```
            do i = 1 to court.mandatory.num;
               put skip list ("   ", court.mandatory.ct(i));
               end;
            put skip list ("What court number would you like to add ? ");
            mandatory.num = mandatory.num + 1;
            get list (court.mandatory.ct(mandatory.num));
            put skip list ("Do you want to add more ? (y/n) ");
            get list (ans);
            end;        /* do while */
         end;           /* else do */

      end;    /* then do (if mand.num > 0 ) */

   else do;          /* no mandatory courts */
      put skip list ("Change weights for preferred courts ? (y/n) ");
      get list (ans);
      if ans = "y"
      then do i = 1 to num_courts;
         put skip edit ("Enter the i.d. number of court number ",i,
                        " in preferential order ")  (a, f(2,0), a);
         get list (league.court.pref.ct(i));
         court_list(i) = league.court.pref.ct(i);
         put skip list ("What is the weight for this court ? ");
         get list (league.court.pref.wt(i));
         end;        /* then do i */
      end;         /* else do (no mandatory courts) */

   end;       /* relax */
```

```
printout: procedure (misc_ptr, season_ptr, div_ptr, div_s_ptr,
                                sched_ptr, lg, div);

%include structures;
%include div_dcl;
%include sched_dcl;
declare sysin file stream input;
declare sysprint file stream output;

declare 1 team_sched(8,14),
          2 opponent fixed,
          2 d_o_w char(9) varying,
          2 mon char(9) varying,
          2 idate fixed,
          2 ct_name char(8) varying,
          2 time fixed;

declare 1 ts_string(15),
          2 hr fixed,
          2 mins char(4);

declare (i, iday, lg, div, t1, t2, count(8), hr, t, cts) fixed;
declare min char(4);
declare time fixed decimal(3,1);
declare ans char(1);
declare day_name(7) char(9) varying initial("Sunday", "Monday",
          "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday");

/* set up time-slot array for output */
time = misc.time_slots.start;
do i = 1 to misc.time_slots.num;
   ts_string(i).hr = floor(time);
   if time - floor(time) = 0.0
   then ts_string(i).mins = ":00 ";
   else ts_string(i).mins = ":30 ";
   time = time + misc.time_slots.length;
   end;

put skip list ("************************************************");
put skip edit ("Schedule for ",misc.league(lg).name,"-league division ",
                     div) (3 a, f(2,0));
put skip list ("Teams are:");
do i = 1 to division.num_teams;
   put skip edit (division.team_number(i), "   ", team_name(i))
                                          (f(8,0), 2 a);
   end;
put skip;

do i = 1 to 8;
   count(i) = 0;
   end;

do i = 1 to divsched.n_games;
   t1 = divsched.game(i).t11;
   t2 = divsched.game(i).t22;
   count(t1) = count(t1) + 1;
   count(t2) = count(t2) + 1;
   iday = divsched.game(i).day;
   team_sched(t1,count(t1)).opponent = team_number(t2);
   team_sched(t2,count(t2)).opponent = team_number(t1);
```

```
      team_sched(t1,count(t1)).d_o_w=day_name(season.day(iday).day_of_wk);
      team_sched(t2,count(t2)).d_o_w=day_name(season.day(iday).day_of_wk);
      team_sched(t1,count(t1)).mon=misc.month(season.day(iday).month).name;
      team_sched(t2,count(t2)).mon=misc.month(season.day(iday).month).name;
      team_sched(t1,count(t1)).idate = season.day(iday).date;
      team_sched(t2,count(t2)).idate = season.day(iday).date;
      team_sched(t1,count(t1)).ct_name = court(divsched.game(i).ct).name;
      team_sched(t2,count(t2)).ct_name = court(divsched.game(i).ct).name;
      team_sched(t1,count(t1)).time = divsched.game(i).time;
      team_sched(t2,count(t2)).time = divsched.game(i).time;
      hr = ts_string(divsched.game(i).time).hr;
      min = ts_string(divsched.game(i).time).mins;
      put skip edit(division.team_number(t1)," plays ",
            division.team_number(t2), hr, min, team_sched(t1,count(t1)).
            d_o_w, " ", " ",team_sched(t1,count(t1)).mon,
            team_sched(t1,count(t1)).idate," on ",
            team_sched(t1,count(t1)).ct_name)
                  (f(3,0), a, 2 f(3,0), 4 a, f(3,0), 2 a);
   end;   /* do i */

put skip list ("Would you like schedule listings by team? (y/n)");
get list (ans);
if ans = "y"
then do t = 1 to division.num_teams;
   put skip edit ("Schedule of team ",team_number(t)," : ",
                  team_name(t))  (a, f(3,0), 2 a);
   do i = 1 to count(t);
      t1 = team_sched(t,i).time;
      put skip edit ("  Plays ",team_sched(t,i).opponent,
            ts_string(t1).hr,ts_string(t1).mins,team_sched(t,i).d_o_w,
            ", ",team_sched(t,i).mon,team_sched(t,i).idate, " on ",
            team_sched(t,i).ct_name) (a,2 f(3,0),4 a,f(3,0),2 a);
      end;           /* do i */
   end;      /* do t */

cts = misc.num_courts;
put skip list("Would you like to see the full schedule? (y/n)");
get list (ans);
if ans = "n" then return;

put skip edit (" ", court(1).name) (a(19), a(10));
do i = 2 to cts;
   put edit (court(i).name)  (a(10));
   end;
put skip;

do iday = 1 to season.num_days;
   put skip edit(day_name(season.day(iday).day_of_wk), ts_string(1).hr,
         ts_string(1).mins)  (a(12), f(3,0), a);
   do i = 1 to cts;
      put edit ( sched(iday,i,1).t11, "-", sched(iday,i,1).t22, "")
                        (f(4,0), a, f(3,0), a(2));
      end;           /* do i */
   put skip edit (misc.month(season.day(iday).month).name,
            season.day(iday).date, ts_string(2).hr, ts_string(2).mins)
            (a(9), 2 f(3,0), a);
   do i = 1 to cts;
      put edit (sched(iday,i,2).t11, "-", sched(iday,i,2).t22, "")
                        (f(4,0), a, f(3,0), a(2));
      end;           /* do i */
```

```
      do t = 3 to misc.time_slots.num;
         put skip edit(" ", ts_string(t).hr, ts_string(t).mins)
                                              (a(12), f(3,0), a);
         do i = 1 to cts;
             put edit (sched(iday,i,t).t11, "-", sched(iday,i,t).t22, "")
                                  (f(4,0), a, f(3,0), a(2));
             end;        /* do i */
         end;            /* do t */

      put skip;
      end;       /* do iday */

   return;
   end;       /* printout */
```

INTRAMURAL SPORTS SCHEDULING PROGRAM

USER'S HANDBOOK

INTRODUCTION

Every effort has been made to make this program as powerful and easy to use as possible. It should greatly simplify the scheduling task for managers who understand how to use it to its fullest capabilities. However, without a careful reading of this handbook and a certain amount of advanced preparation prior to sitting down at the terminal, the manager may find himself facing an array of puzzling problems. To insure a quick and successful terminal session the user should do the following:

1. Read the rest of this manual carefully.
2. Do the advanced preparation suggested in the following paragraphs.
3. Be very careful to input all data correctly when at the terminal.
4. Be very careful to follow the specified format for inputting each data item.

This last item refers to such issues as whether the program wants a number to be input as an integer (11 for instance) or a fraction (11.0). In most cases the program specifies the format needed, but in general the following rules apply.

A. If a number is requested enter it as an integer unless the number represents a time of day.

B. All times are to be entered as decimal fractions representing the number of hours on a 24-hour clock. (Thus 9 a.m. = 9.0, 4:30 p.m. = 16.5, noon = 12.0, midnight = 0.0)

C. When a day of the week is to be entered, an integer representing that day's position in the week should be used. (i.e. Sunday=1, Tuesday = 3, Friday = 6, etc.)

D. When a month is asked for, an integer should be entered that represents that month's position in the list of months comprising

the season. (i.e. if a 4 month season starts in May, June = 2 not 6)

SPECIAL FEATURES

A number of optional features have been included in this program to give the manager various forms of control over the "shape" of the resulting schedule. The manager can seed the teams within each division causing the best teams to meet in the last week of the season; provide specific days and/or times on specific days that particular teams cannot play; specify a set of mandatory courts for a league; specify a weighted preferential order for choosing courts to use for a league; and specify a preferential order for choosing courts to use for a league; and specify a preferential weighting of days of the week and times of day to be used. Each of these capabilities is described seperately in the paragraphs that follow. The manager should determine which of these capabilities he desires to use and do any advanced preparation suggested in that section.

SEEDING - The program is designed to schedule each division so that teams 1 and 2 play in the last week, 1 and 3 play the week before, and so on. This provides the ability to have the strongest teams meet toward the end of the regular season if that is desired. If so, the manager should simply enter the teams in order of strength, with the strongest entered first. If seeding is not desired, the teams can be entered in some random order. How strength of teams is determined is up to the manager -- presumably he has some hunches along that line.

TEAM TIME-CONSTRAINTS - Often, teams will be unable to play on certain dates in the season and specify this on their entry card. When each division is about to be scheduled, the manager will be asked to enter the teams in that division and dates on which any of them cannot play. The manager should include on his list of teams, a list beside each team of dates that team cannot play. These dates should include the month index discussed in part D of the Introduction.

Occasionally, a team will be unable to play on only a portion of some particular date. In that case the manager can specify the month index and date involved and the indices of the first and last time-slots in the unavailable portion of that day. This information should be written down on the team list before sitting down to the terminal.

LEAGUE-WIDE CONSTRAINTS - The following constraints are entered once for each league and are applied to all of the divisions within that league.

A. Mandatory Courts - If some of the courts are unsuitable for A-league (or even B) play, the manager can specify one or more courts that must be used when scheduling the league at hand.

B. Preferred Courts - If the manager does not want to absolutely require the league's games to be scheduled on a particular court(s) but still would prefer to see some courts used more often than others, he can list the courts in preferential order and give a weight to each one. The higher the weight, the harder the program tries to satisfy that preference.

The functioning of these weights is basically as follows. In general the program schedules each game in the first open slot it finds that satisfies the mandatory constraints (court open, both teams can play then, mandatory court (if any) is satisfied) and provides the requested minimum spacing between a team's consecutive games. If a preferential constraint has been specified, and the first open slot found does not satisfy that

preference, the program will search through the season in an attempt to find a slot that does. The number of days it will search is the number specified by the weight. It is important not to set these weights too high (8 is a good limit to keep, 15 is probably an absolute maximum), because if they are set too high the individual blocks of games within a division may become so spread out that all of the blocks cannot be fit into the season while still preserving the requested minimum spacing between blocks. (A division of 6 teams playing 5 games each will have 5 "blocks" of games scheduled, each block containing 3 games, so that each team plays once in each block) In addition to limiting the preference seeking search, the weights' values also rate the relative importance of different preferential constraints. Often, it may be possible to satisfy either of a pair of preferential constraints, but not both. In that case, the program will satisfy the one weighted more highly.

C. Preferred Day of Week - The user may want to schedule games for a league on a particular day of the week whenever possible. To facilitate this, he may list up to 5 preferred days and weight each one. It is important that the most highly preferred day be entered first. The user should draw up this list of days (day numbers) and associated weights ahead of time.

D. Preferred Time-Slots - If the user wishes to specify a particular block of time slots on a certain day of the week as preferred he may do this also. He can list up to 5 such blocks, and should list the most preferred first. For each group of times he will need to specify the index for the day of the week this preferred group of slots falls on, the indices of the first and last time-slots in this block, and the weight

for the block.

GENERAL PREPARATION

The following items should be prepared before sitting down at the terminal, regardless of what optional features are to be used. The time spent creating the lists suggested will be regained through a much shorter and less frustating session at the terminal.

1. Calendar - The user first needs to determine what days will be included in the season. To communicate this information to the computer he will provide information about the months that the season spans and tell it whether all of the days in that period should be included, or just the weekends or weekdays. This provides the basic calendar, but does not take into account exceptions such as holidays which should be removed from the calendar. To provide for this, the program asks for days that would normally be part of the season (i.e. Saturdays and Sundays if it is a weekend sport) but are to be removed. The user should list these ahead of time by month index and date. (i.e. for a September to November sport, October 12 = 2, 12)

2. Time-Slots - One set of time slots is input that will be used for every day in the season. Thus, the user must figure out the earliest time that a game will start on any day in the season, and the latest time for any day in the season and enter these limits. What the program actually asks for is the start of the earliest slot, the end of the latest slot, and the length of each slot. While the user only needs to enter these limiting values, he should list all of the slots' starting times both to

check this against the list the computer produces and to aid in determining the time-slots that each facility is available.

3. Facility Availabilities - To determine the slots available in the season the program asks for a sample week of availabilities and replicates this through the season using its internal calendar. Exceptions to these typical availabilities can then be entered to take care of days on which varsity meets will be occupying one of the courts and so on.

The sample week should contain the most typical availabilities, as time slots can be added or subtracted from this for dates that are entered as exceptions. For each court the user should determine what time slots will typically be available on each day of the week. A string of N digits should then be generated, where N is the number of time slots known to the program. The first of these digits will represent the first time slot and so on. For each time-slot that is available on that court on that day the user should put a 1 as the corresponding digit. For each unavailable slot he should put a 0. This N-digit "bit-string" will then contain as many ones as there are available slots on that court on that day, and their positioning tells which slots are the open ones.

Specific dates in the season on which one of the courts has more or fewer open slots than usual are then entered as exceptions. The user should list ahead of time the exceptions that will need to be entered, providing for each the month index, date, court index, and the new bit-string to be used.

5. Division Data - For each of the divisions the manager will need to determine the minimum number of games that each team will play, and the minimum number of days to be skipped between the end of one block of games

and the start of the next. This minimum spacing ensures that consecutive games of any team in the division will be seperated by at least that many days. It is nice to keep this space large so that a team's games are spread evenly over the course of the entire season, but if it is made too large, the program will be unable to schedule the division. If this happens, however, the program will allow the user to specify a new, smaller spacing and try again to schedule the division. Thus, the user may want to try requesting a large spacing first and then back down on it until the scheduling is successful.

With this preparation done ahead of time you should find that the program will save time in scheduling and provide a schedule that meets with all of your wishes.

APPENDIX D

SAMPLE RUNS

```
Welcome to the intramural sports scheduling system.
Used, properly this program should be of great assistance but you may
run into problems if you have not first looked over the user's manual.
If you have not yet done so, you are advised to stop
here and do some reading.  Do you wish to continue? (y/n) y

Please be careful to always enter data in the format
specified by the question.  For instance, when responses are given
in brackets at the end of the question, choose one of
them.  What is the sport name ?   Basketball

How many leagues (i.e. A,B,C, etc.) will there be ?  2

You will now be asked for the name of each league and
the number of divisions in each.  Please input the
leagues in the order you want them scheduled in. Also,
remember that divisions must have 4 to 8 teams in them
What is the name of league              1          ?   A

How many divisions in league            1          ?  2

What is the name of league              2          ?   B

How many divisions in league            2          ?  2

Is sport a weekend(1), weekday(2), or full-week(3) sport? 1

How many months does the regular season span ?  3

What is the name of month 1 ? June

What day of the week does this month start on?
Please give a number, i.e. Sunday=1, Friday=6, etc.  3

How many days in this month ? i.e. 28,29,30,or 31   30

What is the name of month 2 ? July

What day of the week does this month start on?
Please give a number, i.e. Sunday=1, Friday=6, etc.  5

How many days in this month ? i.e. 28,29,30,or 31   31

What is the name of month 3 ? August

What day of the week does this month start on?
Please give a number, i.e. Sunday=1, Friday=6, etc.  1

How many days in this month ? i.e. 28,29,30,or 31   31

What date will season start on? (i.e. the first day in June that's in season)

What date will season end on ? (no month, just date )22

How many courts are available ?  3

How long are the time-slots for each game?  Please respond with a fraction
representing the number of hours. (i.e. 90 minutes = 1.5  1.5

When does the earliest time slot (on any day) start?
      [9:30 a.m. = 9.5,  1:00 p.m. = 13.0]  11.0
```

```
When does the latest time-slot (on any day) end?  17.0


Please enter the names of the courts in their
default order of preference.
What is the name of facility           1          ? (8 chars max) Case

What is the name of facility           2          ? (8 chars max) Dupont1

What is the name of facility           3          ? (8 chars max) Dupont2

Time-slot  1 is 11:00 a.m.
Time-slot  2 is 12:30 p.m.
Time-slot  3 is  2:00 p.m.
Time-slot  4 is  3:30 p.m.
You will be asked to enter the availabilities of each of the courts
for the 'typical' days in the season.  Please enter these availabilities as
 4-bit numbers with a 1 for each availabletime-slot
and a 0 for each unavailable slot
Please enter the availability of court 'Case' on a typical Sunday 1111

Please enter the availability of court 'Dupont1' on a typical Sunday 1111

Please enter the availability of court 'Dupont2' on a typical Sunday 1111

Please enter the availability of court 'Case' on a typical Saturday 1000

Please enter the availability of court 'Dupont1' on a typical Saturday 1111

Please enter the availability of court 'Dupont2' on a typical Saturday 1111

Were all strings entered correctly ? (y/n) n

Enter day of week to be changed as a number  7

Enter court number to be changed  1

What is the correct availability string ?  1111

Is everything correct now ?  y


You will now be asked for dates (e.g. Holidays)
which fall on days normally part of the season, but
on which no games will be played.  Note that this
does not include days on which only some of the
courts are unavailable.
Are there any unavailable days in       June       ? n

Are there any unavailable days in       July       ? y

What is the date ? 4

Any more unavailable days in  July       ? (y/n) n

Are there any unavailable days in       August     ? n


Are there any days on which more or fewer time
slots than usual will be open ? (y/n) n


We are now ready to schedule A-league
Is there a mandatory court (or courts) for this league? (y/n) y
```

```
How many ?  1

Please input the numbers of these courts in preferential order
Court  1 ? 1

Are there any preferred days of the week for schedulinsA-leasue sames ? (y/n) y

How many preferred days are there ?  1

Please enter these preferred days in order of desirability.  Alons with
each day enter a weishtins factor correspondins to the number of days you
would be willins to delay the schedulins of a particular same to make
it fall on that day.
Day  1 ? 1

Weishtins factor ?  5

Are there any times that are preferred for schedulinsA-leasue sames ? y

Preferred times consist of a block of time-slots on a particular day of
the week.  You may specify up to 5 such blocks by sivins the day, startins
slot and endins slot of each prefered block.  You will also be asked to sive
a weisht to each block correspondins to the number of days a same's
schedulins may be delayed in order to fall into that block.
How many such blocks do you wish to specify ?  1

Enter the number correspondins to the day on which
block  1(in preferential order) occurs. 1

Enter the number of the time-slot which besins this block.3

Enter the number of the last time-slot in block. 4

Weisht for this block ?  5


We are now ready to schedule division number                1
        of A-leasue
How many teams in this division? (must be 4 to 8)  6

How many sames will each team play?  5

What is the name of team 1 ? pru_dos

What is the name of team 2 ? bob

What is the name of team 3 ? 'los

What is the name of team 4 ? maria

What is the name of team 5 ? peter

What is the name of team 6 ? eric

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  n

How many days would you like to leave between
    consecutive sames for each team in this division  10
```

```
**********************************************
Schedule for A-league division  1
Teams are:
       1   pru_dos
       2   bob
       3   'los
       4   maria
       5   peter
       6   eric

  3 plays   4 14:00 Sunday, June 13 on Case
  2 plays   5 15:30 Sunday, June 13 on Case
  1 plays   6 11:00 Sunday, June 13 on Case
  3 plays   6 14:00 Sunday, June 27 on Case
  2 plays   4 15:30 Sunday, June 27 on Case
  1 plays   5 11:00 Sunday, June 27 on Case
  5 plays   6 14:00 Sunday, July 11 on Case
  2 plays   3 15:30 Sunday, July 11 on Case
  1 plays   4 11:00 Sunday, July 11 on Case
  4 plays   5 14:00 Sunday, July 25 on Case
  2 plays   6 15:30 Sunday, July 25 on Case
  1 plays   3 11:00 Sunday, July 25 on Case
  3 plays   5 14:00 Sunday, August  8 on Case
  4 plays   6 15:30 Sunday, August  8 on Case
  1 plays   2 11:00 Sunday, August  8 on Case
Would you like schedule listings by team? (y/n) y

Schedule of team   1 : pru_dos
  Plays   6 11:00 Sunday, June 13 on Case
  Plays   5 11:00 Sunday, June 27 on Case
  Plays   4 11:00 Sunday, July 11 on Case
  Plays   3 11:00 Sunday, July 25 on Case
  Plays   2 11:00 Sunday, August  8 on Case
Schedule of team   2 : bob
  Plays   5 15:30 Sunday, June 13 on Case
  Plays   4 15:30 Sunday, June 27 on Case
  Plays   3 15:30 Sunday, July 11 on Case
  Plays   6 15:30 Sunday, July 25 on Case
  Plays   1 11:00 Sunday, August  8 on Case
Schedule of team   3 : 'los
  Plays   4 14:00 Sunday, June 13 on Case
  Plays   6 14:00 Sunday, June 27 on Case
  Plays   2 15:30 Sunday, July 11 on Case
  Plays   1 11:00 Sunday, July 25 on Case
  Plays   5 14:00 Sunday, August  8 on Case
Schedule of team   4 : maria
  Plays   3 14:00 Sunday, June 13 on Case
  Plays   2 15:30 Sunday, June 27 on Case
  Plays   1 11:00 Sunday, July 11 on Case
  Plays   5 14:00 Sunday, July 25 on Case
  Plays   6 15:30 Sunday, August  8 on Case
Schedule of team   5 : peter
  Plays   2 15:30 Sunday, June 13 on Case
  Plays   1 11:00 Sunday, June 27 on Case
  Plays   6 14:00 Sunday, July 11 on Case
  Plays   4 14:00 Sunday, July 25 on Case
  Plays   3 14:00 Sunday, August  8 on Case
Schedule of team   6 : eric
  Plays   1 11:00 Sunday, June 13 on Case
  Plays   3 14:00 Sunday, June 27 on Case
  Plays   5 14:00 Sunday, July 11 on Case
  Plays   2 15:30 Sunday, July 25 on Case
  Plays   4 15:30 Sunday, August  8 on Case
Would you like to see the full schedule? (y/n) n
```

```
We are now ready to schedule division number              2
        of A-league
How many teams in this division? (must be 4 to 8)  4

How many games will each team play?  6

What is the name of team 1 ? marv

What is the name of team 2 ? Jody

What is the name of team 3 ? brenda

What is the name of team 4 ? mark

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  n

How many days would you like to leave between
    consecutive games for each team in this division  10

**********************************************
Schedule for A-league division  2
Teams are:
       7   marv
       8   Jody
       9   brenda
      10   mark

  8 plays   9 12:30 Sunday, June 13 on Case
  7 plays  10 11:00 Saturday, June 12 on Case
  8 plays  10 12:30 Sunday, June 27 on Case
  7 plays   9 11:00 Saturday, June 26 on Case
  9 plays  10 12:30 Sunday, July 11 on Case
  7 plays   8 11:00 Saturday, July 10 on Case
  8 plays   9 12:30 Sunday, July 25 on Case
  7 plays  10 11:00 Saturday, July 24 on Case
  8 plays  10 12:30 Sunday, August  8 on Case
  7 plays   9 11:00 Saturday, August  7 on Case
  9 plays  10 14:00 Sunday, August 22 on Case
  7 plays   8 15:30 Sunday, August 22 on Case
Would you like schedule listings by team? (y/n) n

Would you like to see the full schedule? (y/n) y

                      Case       Dupont1    Dupont2

Saturday      11:00    7- 10      0-  0      0-  0
June       12 12:30    0-  0      0-  0      0-  0
              14:00    0-  0      0-  0      0-  0
              15:30    0-  0      0-  0      0-  0

Sunday        11:00    1-  6      0-  0      0-  0
June       13 12:30    8-  9      0-  0      0-  0
              14:00    3-  4      0-  0      0-  0
              15:30    2-  5      0-  0      0-  0

Saturday      11:00    0-  0      0-  0      0-  0
June       19 12:30    0-  0      0-  0      0-  0
              14:00    0-  0      0-  0      0-  0
              15:30    0-  0      0-  0      0-  0
```

Sunday		11:00	0-	0	0-	0	0-	0
June	20	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Saturday		11:00	7-	9	0-	0	0-	0
June	26	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	1-	5	0-	0	0-	0
June	27	12:30	8-	10	0-	0	0-	0
		14:00	3-	6	0-	0	0-	0
		15:30	2-	4	0-	0	0-	0
Saturday		11:00	0-	0	0-	0	0-	0
July	3	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	0-	0	0-	0	0-	0
July	4	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Saturday		11:00	7-	8	0-	0	0-	0
July	10	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	1-	4	0-	0	0-	0
July	11	12:30	9-	10	0-	0	0-	0
		14:00	5-	6	0-	0	0-	0
		15:30	2-	3	0-	0	0-	0
Saturday		11:00	0-	0	0-	0	0-	0
July	17	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	0-	0	0-	0	0-	0
July	18	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Saturday		11:00	7-	10	0-	0	0-	0
July	24	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	1-	3	0-	0	0-	0
July	25	12:30	8-	9	0-	0	0-	0
		14:00	4-	5	0-	0	0-	0
		15:30	2-	6	0-	0	0-	0
Saturday		11:00	0-	0	0-	0	0-	0
July	31	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	0-	0	0-	0	0-	0
August	1	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Saturday		11:00	7-	9	0-	0	0-	0

August	7	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	1-	2	0-	0	0-	0
August	8	12:30	8-	10	0-	0	0-	0
		14:00	3-	5	0-	0	0-	0
		15:30	4-	6	0-	0	0-	0
Saturday		11:00	0-	0	0-	0	0-	0
August	14	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	0-	0	0-	0	0-	0
August	15	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Saturday		11:00	0-	0	0-	0	0-	0
August	21	12:30	0-	0	0-	0	0-	0
		14:00	0-	0	0-	0	0-	0
		15:30	0-	0	0-	0	0-	0
Sunday		11:00	0-	0	0-	0	0-	0
August	22	12:30	0-	0	0-	0	0-	0
		14:00	9-	10	0-	0	0-	0
		15:30	7-	8	0-	0	0-	0

We are now ready to schedule B-league
Is there a mandatory court (or courts) for this league? (y/n) n

Are there preferred courts for this league? (If not, the order in which the court names were originally entered will be used as the default.) y

Enter the court numbers in order of desirability. Along with each court number please enter a weighting factor reflecting how desirable that court is. The weight represents the number of days that you are willing to push a game back in the schedule in order to have it scheduled on that court.
Court 1 ? 1

What is the weight for this court ? 5

Court 2 ? 2

What is the weight for this court ? 1

Court 3 ? 3

What is the weight for this court ? 1

Are there any preferred days of the week for schedulingB-league games ? (y/n)

How many preferred days are there ? 1

Please enter these preferred days in order of desirability. Along with each day enter a weighting factor corresponding to the number of days you would be willing to delay the scheduling of a particular game to make it fall on that day.
Day 1 ? 1

Weighting factor ? 2

```
Are there any times that are preferred for schedulinsB-leasue sames ? n


We are now ready to schedule division number                1
        of B-leasue
How many teams in this division? (must be 4 to 8)  7

How many sames will each team play?  5

What is the name of team 1 ? tano

What is the name of team 2 ? is

What is the name of team 3 ? crazy

What is the name of team 4 ? hope

What is the name of team 5 ? izza

What is the name of team 6 ? srape

What is the name of team 7 ? iiad_ibvh

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  y

What team? (enter its number)  1

What month and date is         tano
       not able to play on? (enter as numbers)  1 12

Are there more days on which teams in this
          division cannot play? (y/n)  y

What team? (enter its number)  1

What month and date is         tano
       not able to play on? (enter as numbers)  1 13

Are there more days on which teams in this
          division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  n

How many days would you like to leave between
    consecutive sames for each team in this division  5

*****************************************
Schedule for B-leasue division  1
Teams are:
      11   tano
      12   is
      13   crazy
      14   hope
      15   izza
      16   srape
      17   iiad_ibvh

 13 plays  14 12:30 Saturday, June 12 on Case
 12 plays  15 14:00 Saturday, June 12 on Case
 11 plays  16 11:00 Sunday, June 20 on Case
```

```
 16 Plays  17 12:30 Saturday, June 26 on Case
 12 Plays  14 14:00 Saturday, June 26 on Case
 11 Plays  15 15:30 Saturday, June 26 on Case
 15 Plays  17 11:00 Saturday, July  3 on Case
 12 Plays  13 12:30 Saturday, July  3 on Case
 11 Plays  14 14:00 Saturday, July  3 on Case
 14 Plays  17 12:30 Saturday, July 10 on Case
 15 Plays  16 14:00 Saturday, July 10 on Case
 11 Plays  13 15:30 Saturday, July 10 on Case
 14 Plays  16 11:00 Sunday, July 18 on Case
 13 Plays  17 12:30 Sunday, July 18 on Case
 11 Plays  12 14:00 Sunday, July 18 on Case
 14 Plays  15 12:30 Saturday, July 24 on Case
 13 Plays  16 14:00 Saturday, July 24 on Case
 12 Plays  17 15:30 Saturday, July 24 on Case
Would you like schedule listings by team? (y/n) n

Would you like to see the full schedule? (y/n) n


We are now ready to schedule division number              2
       of B-league
How many teams in this division? (must be 4 to 8)  6

How many games will each team play?  3

What is the name of team 1 ? Jerry

What is the name of team 2 ? bob

What is the name of team 3 ? Phil

What is the name of team 4 ? bill

What is the name of team 5 ? mickey

What is the name of team 6 ? brent

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  y

What team? (enter number)  1

What month and date? (enter month as number)  1 12

Now enter the numbers corresponding to the first
    and last time-slots in the unavailable block  1 3

Are there more blocks of time slots on which teams
       in this division cannot play? (y/n)  y

What team? (enter number)  1

What month and date? (enter month as number)  1 13

Now enter the numbers corresponding to the first
    and last time-slots in the unavailable block  1 3

Are there more blocks of time slots on which teams
       in this division cannot play? (y/n)  n

How many days would you like to leave between
```

```
    consecutive games for each team in this division  8/

 Scheduling of division was unsuccessful with a
min-space of 87Would you like more info? n

Would you like to rerun with a new min-space? (y/n)  y

Change weights for preferred courts ? (y/n)  n

How many days would you like to leave between
    consecutive games for each team in this division  4

**********************************************
Schedule for B-league division  2
Teams are:
      18   Jerry
      19   bob
      20   Phil
      21   bill
      22   mickey
      23   brent

 22 plays  23 15:30 Saturday, June 12 on Case
 19 plays  20 11:00 Sunday, June 13 on Dupont1
 18 plays  21 15:30 Sunday, June 13 on Dupont1
 21 plays  22 12:30 Sunday, June 20 on Case
 19 plays  23 14:00 Sunday, June 20 on Case
 18 plays  20 15:30 Sunday, June 20 on Case
 20 plays  22 11:00 Sunday, June 27 on Dupont1
 21 plays  23 12:30 Sunday, June 27 on Dupont1
 18 plays  19 14:00 Sunday, June 27 on Dupont1
Would you like schedule listings by team? (y/n) n

Would you like to see the full schedule? (y/n) y

                      Case       Dupont1    Dupont2

Saturday        11:00    7- 10      0-  0      0-  0
June       12   12:30   13- 14      0-  0      0-  0
                14:00   12- 15      0-  0      0-  0
                15:30   22- 23      0-  0      0-  0

Sunday          11:00    1-  6     19- 20      0-  0
June       13   12:30    8-  9      0-  0      0-  0
                14:00    3-  4      0-  0      0-  0
                15:30    2-  5     18- 21      0-  0

Saturday        11:00    0-  0      0-  0      0-  0
June       19   12:30    0-  0      0-  0      0-  0
                14:00    0-  0      0-  0      0-  0
                15:30    0-  0      0-  0      0-  0

Sunday          11:00   11- 16      0-  0      0-  0
June       20   12:30   21- 22      0-  0      0-  0
                14:00   19- 23      0-  0      0-  0
                15:30   18- 20      0-  0      0-  0

Saturday        11:00    7-  9      0-  0      0-  0
June       26   12:30   16- 17      0-  0      0-  0
                14:00   12- 14      0-  0      0-  0
                15:30   11- 15      0-  0      0-  0

Sunday          11:00    1-  5     20- 22      0-  0
June       27   12:30    8- 10     21- 23      0-  0
                14:00    3-  6     18- 19      0-  0
                15:30    2-  4      0-  0      0-  0
```

Saturday		11:00	15- 17	0- 0	0- 0	
July	3	12:30	12- 13	0- 0	0- 0	
		14:00	11- 14	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Sunday		11:00	0- 0	0- 0	0- 0	
July	4	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Saturday		11:00	7- 8	0- 0	0- 0	
July	10	12:30	14- 17	0- 0	0- 0	
		14:00	15- 16	0- 0	0- 0	
		15:30	11- 13	0- 0	0- 0	
Sunday		11:00	1- 4	0- 0	0- 0	
July	11	12:30	9- 10	0- 0	0- 0	
		14:00	5- 6	0- 0	0- 0	
		15:30	2- 3	0- 0	0- 0	
Saturday		11:00	0- 0	0- 0	0- 0	
July	17	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Sunday		11:00	14- 16	0- 0	0- 0	
July	18	12:30	13- 17	0- 0	0- 0	
		14:00	11- 12	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Saturday		11:00	7- 10	0- 0	0- 0	
July	24	12:30	14- 15	0- 0	0- 0	
		14:00	13- 16	0- 0	0- 0	
		15:30	12- 17	0- 0	0- 0	
Sunday		11:00	1- 3	0- 0	0- 0	
July	25	12:30	8- 9	0- 0	0- 0	
		14:00	4- 5	0- 0	0- 0	
		15:30	2- 6	0- 0	0- 0	
Saturday		11:00	0- 0	0- 0	0- 0	
July	31	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Sunday		11:00	0- 0	0- 0	0- 0	
August	1	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Saturday		11:00	7- 9	0- 0	0- 0	
August	7	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	
Sunday		11:00	1- 2	0- 0	0- 0	
August	8	12:30	8- 10	0- 0	0- 0	
		14:00	3- 5	0- 0	0- 0	
		15:30	4- 6	0- 0	0- 0	
Saturday		11:00	0- 0	0- 0	0- 0	
August	14	12:30	0- 0	0- 0	0- 0	
		14:00	0- 0	0- 0	0- 0	
		15:30	0- 0	0- 0	0- 0	

Sunday		11:00	0-	0	0-	0	0-	0	
August	15	12:30	0-	0	0-	0	0-	0	111
		14:00	0-	0	0-	0	0-	0	
		15:30	0-	0	0-	0	0-	0	
Saturday		11:00	0-	0	0-	0	0-	0	
August	21	12:30	0-	0	0-	0	0-	0	
		14:00	0-	0	0-	0	0-	0	
		15:30	0-	0	0-	0	0-	0	
Sunday		11:00	0-	0	0-	0	0-	0	
August	22	12:30	0-	0	0-	0	0-	0	
		14:00	9-	10	0-	0	0-	0	
		15:30	7-	8	0-	0	0-	0	

r 04:02 10.091 138

RUN 2

```
Welcome to the intramural sports scheduling system.
Used, properly this program should be of great assistance but you may
run into problems if you have not first looked over the user's manual.
If you have not yet done so, you are advised to stop
here and do some reading.  Do you wish to continue? (y/n) y

Please be careful to always enter data in the format
specified by the question.  For instance, when responses are given
in brackets at the end of the question, choose one of
them.  What is the sport name ?   foosball

How many leagues (i.e. A,B,C, etc.) will there be ?  2

You will now be asked for the name of each league and
the number of divisions in each.  Please input the
leagues in the order you want them scheduled in. Also,
remember that divisions must have 4 to 8 teams in them
What is the name of league                1          ?   Z

How many divisions in league              1          ?  1

What is the name of league                2          ?   Q

How many divisions in league              2          ?  1

Is sport a weekend(1), weekday(2), or full-week(3) sport? 1

How many months does the regular season span ?  1

What is the name of month 1 ? December

What day of the week does this month start on?
Please give a number, i.e. Sunday=1, Friday=6, etc.  2

How many days in this month ? i.e. 28,29,30,or 31   31

What date will season start on? (i.e. the first day in December that's in seaso
n) 6

What date will season end on ? (no month, just date )28

How many courts are available ?  4

How long are the time-slots for each game?  Please respond with a fraction
representing the number of hours. (i.e. 90 minutes = 1.5  3

When does the earliest time slot (on any day) start?
      [9:30 a.m. = 9.5,  1:00 p.m. = 13.0]  10.5

When does the latest time-slot (on any day) end?  22.5


Please enter the names of the courts in their
default order of preference.
What is the name of facility              1          ? (8 chars max) ska

What is the name of facility              2          ? (8 chars max) teb

What is the name of facility              3          ? (8 chars max) oard

What is the name of facility              4          ? (8 chars max) ins
```

```
Time-slot  1 is 10:30 a.m.
Time-slot  2 is  1:30 p.m.
Time-slot  3 is  4:30 p.m.
Time-slot  4 is  7:30 p.m.
You will be asked to enter the availabilities of each of the courts
for the 'typical' days in the season.  Please enter these availabilities as
 4-bit numbers with a 1 for each availabletime-slot
and a 0 for each unavailable slot
Please enter the availability of court 'ska' on a typical Sunday 1110

Please enter the availability of court 'teb' on a typical Sunday 1111

Please enter the availability of court 'oard' on a typical Sunday 1111

Please enter the availability of court 'ins' on a typical Sunday 1111

Please enter the availability of court 'ska' on a typical Saturday 1111

Please enter the availability of court 'teb' on a typical Saturday 1111

Please enter the availability of court 'oard' on a typical Saturday 1111

Please enter the availability of court 'ins' on a typical Saturday 1111

Were all strings entered correctly ? (y/n) y

You will now be asked for dates (e.g. Holidays)
which fall on days normally part of the season, but
on which no games will be played.  Note that this
does not include days on which only some of the
courts are unavailable.
Are there any unavailable days in       December  ? n


Are there any days on which more or fewer time
slots than usual will be open ? (y/n) y

Input the name of the court whose availability
is to be changed. 1#ska

Input the number of the month, and the date to be  modified. 1 7

Please enter the availability of court ska on December  7
as a  4-bit number with a 1for each available time_slot and a 0
for each unavailable slot 1100

Are there other days that have fewer or more
time slots than normal ? (y/n) n


We are now ready to schedule Z-league
Is there a mandatory court (or courts) for this league? (y/n) n

Are there preferred courts for this league?  (If not, the order in which the
court names were originally entered will be used as the default.)  y

Enter the court numbers in order of desirability.  Along with each
court number please enter a weighting factor reflecting how desirable
that court is.  The weight represents the number of days that
you are willing to push a game back in the schedule in order to have
it scheduled on that court.
Court  1 ? 1

What is the weight for this court ?  5
```

```
What is the weisht for this court ?  1

Court  3 ? 3

What is the weisht for this court ?  1

Court  4 ? 4

What is the weisht for this court ?  1

Are there any preferred days of the week for schedulinsZ-leasue sames ? (y/n) n

Are there any times that are preferred for schedulinsZ-leasue sames ? y

Preferred times consist of a block of time-slots on a particular day of
the week.  You may specify up to 5 such blocks by sivins the day, startins
slot and endins slot of each prefered block.  You will also be asked to sive
a weisht to each block correspondins to the number of days a same's
schedulins may be delayed in order to fall into that block.
How many such blocks do you wish to specify ?  1

Enter the number correspondins to the day on which
block  1(in preferential order) occurs. 1

Enter the number of the time-slot which besins this block.3

Enter the number of the last time-slot in block.4

Weisht for this block ?  3


We are now ready to schedule division number                1
        of Z-leasue
How many teams in this division? (must be 4 to 8)  8

How many sames will each team play?  3

What is the name of team 1 ? z1

What is the name of team 2 ? z2

What is the name of team 3 ? z3

What is the name of team 4 ? z4

What is the name of team 5 ? z5

What is the name of team 6 ? z6

What is the name of team 7 ? z7

What is the name of team 8 ? z8

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  n

How many days would you like to leave between
    consecutive sames for each team in this division  5

**********************************************
```

```
Schedule for Z-league division  1
Teams are:
        1   z1
        2   z2
        3   z3
        4   z4
        5   z5
        6   z6
        7   z7
        8   z8

  5 plays   8 10:30 Saturday, December  6 on ska
  6 plays   7 13:30 Saturday, December  6 on ska
  2 plays   3 16:30 Saturday, December  6 on ska
  1 plays   4 19:30 Saturday, December  6 on ska
  4 plays   6 16:30 Sunday, December 14 on ska
  5 plays   7 10:30 Saturday, December 13 on ska
  2 plays   8 13:30 Saturday, December 13 on ska
  1 plays   3 16:30 Saturday, December 13 on ska
  5 plays   6 16:30 Sunday, December 21 on ska
  3 plays   4 10:30 Saturday, December 20 on ska
  7 plays   8 13:30 Saturday, December 20 on ska
  1 plays   2 16:30 Saturday, December 20 on ska
Would you like schedule listings by team? (y/n) n

Would you like to see the full schedule? (y/n) n


We are now ready to schedule Q-league
Is there a mandatory court (or courts) for this league? (y/n) n

Are there preferred courts for this league?  (If not, the order in which the
court names were originally entered will be used as the default.)  y

Enter the court numbers in order of desirability.  Along with each
court number please enter a weighting factor reflecting how desirable
that court is.  The weight represents the number of days that
you are willing to push a game back in the schedule in order to have
it scheduled on that court.
Court  1 ? 1

What is the weight for this court ?  3

Court  2 ? 2

What is the weight for this court ?  0

Court  3 ? 3

What is the weight for this court ?  0

Court  4 ? 4

What is the weight for this court ?  0

Are there any preferred days of the week for schedulingQ-league games ? (y/n) y

How many preferred days are there ?  1

Please enter these preferred days in order of desirability.  Along with
each day enter a weighting factor corresponding to the number of days you
would be willing to delay the scheduling of a particular game to make
it fall on that day.
Day  1 ? 7

Weighting factor ?  2
```

```
Are there any times that are preferred for schedulinsQ-leasue sames ? y

Preferred times consist of a block of time-slots on a particular day of
the week.  You may specify up to 5 such blocks by sivins the day, startins
slot and endins slot of each prefered block.  You will also be asked to sive
a weisht to each block correspondins to the number of days a same's
schedulins may be delayed in order to fall into that block.
How many such blocks do you wish to specify ?  1

Enter the number correspondins to the day on which
block  1(in preferential order) occurs. 7

Enter the number of the time-slot which besins this block.2

Enter the number of the last time-slot in block.3

Weisht for this block ?  5


We are now ready to schedule division number                1
        of Q-leasue
How many teams in this division? (must be 4 to 8)  8

How many sames will each team play?  3

What is the name of team 1 ? q1

What is the name of team 2 ? q2

What is the name of team 3 ? q3

What is the name of team 4 ? q4

What is the name of team 5 ? q5

What is the name of team 6 ? q6

What is the name of team 7 ? q7

What is the name of team 8 ? q8

Are there any days in the season on which one of
the teams in this division cannot play? (y/n)  n


Are there any time_slots (or blocks of slots)
     on which teams in this division cannot play?  n

How many days would you like to leave between
    consecutive sames for each team in this division  5

**********************************************
Schedule for Q-leasue division  1
Teams are:
       9   q1
      10   q2
      11   q3
      12   q4
      13   q5
      14   q6
      15   q7
      16   q8

 13 plays  16 13:30 Saturday, December  6 on teb
 14 plays  15 16:30 Saturday, December  6 on teb
```

```
 10 plays  11 13:30 Saturday, December  6 on oard
  9 plays  12 16:30 Saturday, December  6 on oard
 12 plays  14 13:30 Saturday, December 13 on teb
 13 plays  15 16:30 Saturday, December 13 on teb
 10 plays  16 13:30 Saturday, December 13 on oard
  9 plays  11 16:30 Saturday, December 13 on oard
 13 plays  14 13:30 Saturday, December 20 on teb
 11 plays  12 16:30 Saturday, December 20 on teb
 15 plays  16 13:30 Saturday, December 20 on oard
  9 plays  10 16:30 Saturday, December 20 on oard
Would you like schedule listings by team? (y/n) n

Would you like to see the full schedule? (y/n) y
```

		ska	teb	oard	ins
Saturday	10:30	5- 8	0- 0	0- 0	0- 0
December 6	13:30	6- 7	13- 16	10- 11	0- 0
	16:30	2- 3	14- 15	9- 12	0- 0
	19:30	1- 4	0- 0	0- 0	0- 0
Sunday	10:30	0- 0	0- 0	0- 0	0- 0
December 7	13:30	0- 0	0- 0	0- 0	0- 0
	16:30	0- 0	0- 0	0- 0	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Saturday	10:30	5- 7	0- 0	0- 0	0- 0
December 13	13:30	2- 8	12- 14	10- 16	0- 0
	16:30	1- 3	13- 15	9- 11	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Sunday	10:30	0- 0	0- 0	0- 0	0- 0
December 14	13:30	0- 0	0- 0	0- 0	0- 0
	16:30	4- 6	0- 0	0- 0	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Saturday	10:30	3- 4	0- 0	0- 0	0- 0
December 20	13:30	7- 8	13- 14	15- 16	0- 0
	16:30	1- 2	11- 12	9- 10	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Sunday	10:30	0- 0	0- 0	0- 0	0- 0
December 21	13:30	0- 0	0- 0	0- 0	0- 0
	16:30	5- 6	0- 0	0- 0	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Saturday	10:30	0- 0	0- 0	0- 0	0- 0
December 27	13:30	0- 0	0- 0	0- 0	0- 0
	16:30	0- 0	0- 0	0- 0	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0
Sunday	10:30	0- 0	0- 0	0- 0	0- 0
December 28	13:30	0- 0	0- 0	0- 0	0- 0
	16:30	0- 0	0- 0	0- 0	0- 0
	19:30	0- 0	0- 0	0- 0	0- 0

```
r 04:36 4.555 129
```