A "UNIX" BASED
ELECTRONIC CALENDAR SYSTEM

by

DAVID OWEN JAMES

B. A., Bethany College, Lindsborg, Kansas, 1981

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

[Signature]
Major Professor
ACKNOWLEDGMENT

I wish to thank Dr. Richard A. McBride for his guidance and encouragement throughout the course of this project.
TABLE OF CONTENTS

1. INTRODUCTION ........................................ 1
   1.0 Overview ........................................ 1
   1.1 Rationale of Electronic Calendars ............. 1
   1.2 Scope of the Implementation ................ 3
2. PROBLEM STATEMENT .................................. 5
   2.0 Introduction .................................... 5
   2.1 Requirements ................................... 5
   2.2 Specifications .................................. 7
3. SAMPLE SCENARIO ..................................... 12
   3.0 Introduction ................................... 12
   3.1 Personal Calendar Operations ................ 14
   3.2 Meeting Requests ................................ 22
4. FUTURE ENHANCEMENTS ............................... 28

REFERENCES ........................................... 31

APPENDICIES
   User's Quick Reference ............................. 32
   Design / Maintenance Manual ...................... 35
   Pascal Source Code ................................. 43
CHAPTER 1
INTRODUCTION

1.0 Overview

Appointment calendars are time management tools that allocate specific time segments for activities on a given day. Electronic calendars are computerized versions of the more traditional paper counterparts, but with several advantages, including a wider variety of useful applications. This report contains a description of an implementation of such a system in Pascal on the UNIX operating system currently running on a 32-bit Perkin-Elmer 3220 at the KSU Department of Computer Science.

1.1 Rationale of Electronic Calendars

The obvious advantages of an automated calendar system are the same benefits derived from automating other office processes and transactions: fast access time of computer stored information, storage space efficiency, and automatic (periodic) transaction processing. But many other advantages of automation of appointment schedules exist.

Studies by Mitzberg (1971), and also Boin (1978) have shown that managers and other professionals spend 40-70 percent of the working day in meetings. Costly problems resulting from missed meetings or late arrivals often arise from the fact that individual appointment schedules fall
behind in currency, or that conflicts are inadvertently scheduled during meeting times. If the appointment schedules are manually maintained, random appointment scheduling, such as weekday appointments that don't vary from week to week, can become error prone. Automation in this area could solve many of these problems.

Another advantage of electronic calendar systems can be attributed to appointment scheduling between several principals or users. "Secretaries find scheduling meetings one of the most distasteful aspects of their job, involving many frustrations." (Bancomb, 1981) An Appointment to be made involving several people must be in the intersection of each member's free time schedule, so to speak, but that intersection is not known all at once. "Typically the initial scheduling is done in several passes starting with collection of scheduling constraint information followed by negotiation and selection of a meeting times." (Greif, 1982) A particular time slot that is acceptable to, say, the first three members may not be possible for the fourth. Thus, the process has to be started over, with another attempted proposed meeting time. A study by Bacomb (1981) of this process showed that an average meeting of six people took 60-75 minutes to schedule. In this application, automation is clearly worthwhile. If principal's schedules were kept on computer files instead of disjoint pieces of paper, the "free time intersection" for any number of participants
could easily be calculated. From that information, appointments could easily be set.

1.2 Scope of Implementation

The advantage of electronic calendars systems as cited in the previous sections can be immediately applied to scheduling problems in the KSU Department of Computer Science. Each faculty member keeps some sort of appointment calendar (possibly implicit), to record appointments. Many appointments, such as classes being taught or taken, recur weekly, that is, they are generally the same from week to week. Since the faculty's schedules are often (purposely) staggered, it is often particularly difficult to schedule appointments among them. The automated scheduling algorithms of an electronic calendar system could be a valuable assistance to the administration and students of the department.

This particular implementation of an electronic calendar is designed for use under the UNIX operating system at the KSU Computer Science department. The decision for that choice derives from the fact that UNIX is an industry-wide de facto standard for operating systems on minicomputers, and also current availability of computer resources. There are interactive terminals interfaced to the UNIX system currently operating in most of the offices in the department, so the primary users, the KSU Computer
Science faculty, will have convenient access to the electronic calendar system. A principal, or calendar user, in this application will be defined as any faculty member or graduate student who has possession of a valid UNIX account. Also, a principal could be a non-person, such as a room or other resource, for which daily appointment schedules can be automatically maintained.

The implementation tool is Pascal, specifically, UNIX Berkeley Pascal. (5-7) This choice is, again, primarily based on availability. But it also is a result of the high extent of use of the Pascal language in the department, and the fact that the language and operating system are fairly standard, for portability concerns.
CHAPTER 2
PROBLEM STATEMENT

2.0 Introduction

An electronic calendar system should be written for people to use and benefit from, as a cooperative activity. (Greif, 1982) A priority is ease and convenience in typical daily use. A primary requirement is that the electronic calendar be made available for use interactively 24 hours a day in a multi-user environment. Since all steps performed by the electronic calendar are transmitted to and from the user via interactive CRTs, the level of information displayed to the user should be user-friendly, but concise enough to insure speed and efficiency in use by more experienced users.

2.1 Requirements

Most often, invocation of a calendar program will lead to executing one or two commands, such as looking at tomorrow’s schedule, or requesting a meeting. Therefore, the number of steps required for the user to execute the most common functions be minimal in number, so that the steps can be executed as quickly and as easily as possible.

The more specific requirements related to actual features of a running electronic calendar system can be divided into two major categories. The first is maintenance
of a principal's personal calendar. Maintenance includes insertions and deletions of scheduled appointments. These functions should be applicable to any particular day desired within six months of the current day. A related required feature is that, at the user's option, the insertion or deletion of an appointment should be allowed to apply globally to all subsequent weeks, on the same weekday. As an example, a change for, say, Wednesday, the third of the month, could either only apply to that particular day, or else apply to that particular day and all subsequent Wednesdays, also. In the latter case, the dates affected would be 3, 10, 17, etc. This will be useful for recurring weekly appointments that generally don't change from week to week.

The second category of requirements applies to use of an electronic calendar system in scheduling meetings between two or more principals. A mechanism that allows an appointment request to be sent to all involved principals should be made available to all users of the calendar system. The desire is to only assume confirmation of a requested meeting when all the principals involved have agreed to the request. Each person to whom a request is sent should be notified that the outstanding request does exist, and that an answer is expected. If all requests are answered in the affirmative, the meeting is considered confirmed. If any of the principals rejects the requested
meeting, the meeting is considered canceled, and the members notified, so that possibly another attempt can be made to schedule a meeting.

To both of the previously described categories, some specific requirements apply. Under no circumstances will time conflicts be allowed, that is, an electronic calendar system should always prevent any step executed which would result in overlapping appointments, such as appointment "A" at 8:00 AM - 8:30 AM on the same day as appointment "B" at 8:15 AM - 8:45 AM. Provisions for notifying the user attempting the offending insertion should be made, the particular action to be taken being dependent on the context of the state.

The per-day clock should be one of length 24 hours. Thus, a 2:00 AM - 3:00 AM appointment would be possible. However, normal operation of the calendar should assume an 8:00 AM - 5:00 PM work day, with that default possibly being overridden by the user.

As a final requirement for an electronic calendar system, some provision for backup and purging of old calendar information must be made. The information about past dates does not necessarily need to be directly accessible to the user, but some means for outdated information retrieval must exist.

2.2 Specifications
This section will contain a refinement of the previously defined set of requirements for an electronic calendar system. The refinement is a set of specifications, which precisely defines the outcome of any specific solution to the problem. These specifications do not imply design of the solution, but merely the goals which any answer to the problem must meet.

The electronic calendar system is menu driven; that is, it follows a prompt - and - response format. Clear, concise prompts that require very short responses of the user will be implemented, to assure the speed and efficiency of each execution step at runtime.

In development of the prompts organization and ordering, the list of all possible desired steps executed can be observed. In categorizing the steps, one possibility follows.

1. Observe or edit a particular day's schedule.
2. Observe or edit the events of a typical weekday's schedule which will apply to all succeeding weeks.
3. Send a request to any number of users for a meeting for a particular day.
4. Answer requests made by other users to attend a meeting for a particular day.

While there are clearly other categorizations of steps in an electronic calendar system, such as combining steps one and
two, and/or combining steps three and four, the point to be made here is that each category chosen will entail a different set of prompts, at least to some extent. Therefore a menu-driven format appears to lend itself to this application well. The size of each menu and the number of menus (which are inversely proportional) are determined such that, again, convenience for the user is maximized.

The description of prompts can be most easily conveyed in outline form. Each entry in the following outline will represent a set of one or more prompts required for the transaction to be carried out.

A. Observe or Edit Personal Calendar.

1. For a Particular Day.
   a. Observe.
   b. Insert.
   c. Delete.

2. For a Typical Weekday.
   a. Observe.
   b. Insert.
   c. Delete.

B. Make or Send a Meeting Request.

1. Make a Request.

2. Send a Request.

C. Quit.

Two other requirements also related to prompts are "user friendliness", and convenience. Some specifications
consistent with these requirements can be deduced. The software must be robust. It should accept a wide variety of input as valid answers to prompts. A typical example of this is, as follows: After a prompt for a time of day, either "8:00 AM", "8:00AM", "8:00", or "800" will be accepted. If the program cannot interpret an answer, it will reprompt with a helpful error message, and, of course, the program will never crash as a result of invalid input.

In addition to syntactic input checks, all semantic checks will be performed to be consistent with the "no time conflicts" requirement. This applies any time an insertion attempt is made to any existing schedule.

The electronic calendar system will only make requests for meetings that are scheduled during the "free time intersection" of all principals, in order to prevent time conflicts. However, the option to reject a meeting (presumably for some other reason) is available to any of the principals. In the course of making and answering requests, a more direct communication between users is required than the ones previously described. Therefore, a link to the computer system's interactive mail service will be created to insure currency of users on meeting requests and responses. This process will fulfill the requirements of member notification in scheduling meetings.

A function to backup and purge computer stored information about appointments for past dates will be
provided. The software for this function will be either manually or automatically (periodically) invoked. The details of that process will depend on the current status of available resources of storage devices.
CHAPTER 3
SAMPLE SCENARIO

3.0 Introduction

One way to better understand the implementation of any software product is to observe the operation and results of the program(s) in actual use. For a highly interactive program, such as an electronic calendar, for which prompts - and - responses make up the bulk of its operation, an annotated scenario is of value.

This chapter contains such a scenario which applies specifically to the writer's implementation of an electronic calendar system at the KSU Department of Computer Science. The chapter is intended as a general user's manual, for those users of the UNIX system who desire the facilities provided by the programs of the electronic calendar. However, the objectives implied by this scenario can be applied to electronic calendars in general. It describes a solution to the problem of meeting the formerly defined specifications for any electronic calendar system.

In illustration of the prompts, and the possible responses to those prompts, not all possibilities of input errors will be included in the scope of the chapter. The software is written such that a wide variety of answers to prompts is accepted. This philosophy is consistent with the "user-friendly" requirement. As a specific example; upper
and lower case letters are generally not differentiated — when answering a prompt an alphabetic character may be upper or lower case. The number of "loop until good data" constructs has been kept to a minimum, but there are instances where an escape from a prompt requesting input does not exist. Error messages are helpful in pinning down explanations as to why data is bad. If abnormal termination is desired at any point in the program, a break in the program execution (through the "break" key on most terminals) will not in any way cause harm to the calendar data files used by the program.

The description and illustrations of the sample scenario can be divided into two categories: functions generally dealing with personal appointments, in which the user is the primary principal, and meetings, where possibly several different users of the system share a common calendar entry for a specific date and time. This division is shown in the main menu. As an example, the display seen at invocation of the electronic calendar is:

Welcome to Electronic Calendar

Today is Thu, Nov 18.

Main Menu:
1. Update or Observe Personal Calendar.
2. Make or Answer Meeting Requests.
3. Exit.

Please enter command number or X. ->

The description of the processes contained in choice (1) is discussed in section 3.1, and the processes of choice (2)
are described in section 3.2.

3.1 Personal Calendar Operations

Choosing the first option from the initial main menu puts the user in a mode intended for formatted display and editing of the files contained in that user's account that make up his/her personal calendar. It is assumed that this mode would be used fairly routinely, but for short periods of time at each setting. Therefore the prompts as described here reflect a logical consistency with the order in which decisions would have to be made to observe or edit a manual calendar. The first decision that will have to be made by the user is the date that he/she wants to look at, and possibly edit. The prompt is:

<Ret> for today's schedule, enter day or date. ->

The answer to this prompt is of some importance, in that the scope of the resulting schedule and any edits to that schedule is determined by the user's response. Four possibilities for answers exist: (1) a null line <return>, (2) a date (eg. '24'), (3) a month and date (eg. 'Nov18'), and (4) a weekday (eg. 'Thu'). The scope differences are these. Choices (1) through (3) yield a specific day, in which any changes made only apply to that day. Choice (4) yields a weekday schedule, in which any changes made apply to all succeeding days in the entire calendar which are of the same weekday.
The differences between choices (1), (2), and (3) are minor, the differentiation is primarily a matter of convenience. With the assumption that the most common day to be observed is the current day, a null line entered; that is, an enter (return) key depressed with no previous characters entered, yields the current day's schedule. Choice (2) is equivalent to choice (3), except that the month can only by omitted (as in choice (2)) if the date is within two weeks of the current day.

An example will help illustrate. Suppose that on November 18, 1982 a user wants to observe or edit his personal calendar. Invocation of the electronic calendar system displays the following:

Welcome to Electronic Calendar.

Today is Thu, Nov 18, 1982.

Main Menu:
1. Update or Observe Personal Calendar.
2. Send or Receive Meeting Requests.
X. Exit.

<Ret> for today's schedule, enter day or date. ->

At this point, to observe or edit the current day's schedule (November 18), the user could enter a null line, '18', 'Nov18', or one of several variations of 'Nov18' to yield exactly the same results. The schedule for November 18 will be displayed. If the schedule for, say, Friday, November 19 is to be observed, either '19' or 'Nov19' (or variations) could be entered, with equivalent results. In this example, a single date, without the corresponding month, could be
entered for dates up to and including Wednesday, December 1, which is the two week limit for assumed month names. For any desired date past December 1 (up to a year from the current day), both the month and date must both be given.

To observe a weekday's schedule; that is, the appointments that generally apply to all weeks, the name of the weekday is entered. A three letter abbreviation (eg. 'Thu') is expected, but several variations are accepted. Any changes performed on a weekday schedule will apply to all succeeding weeks.

After successful completion of the date selection process, the schedule for the date (or weekday) is displayed. A heading for the display indicates either the specified day and month, or the general weekday, for which the following schedule applies. The actual schedule follows, and then the list of options which apply to that schedule. A sample schedule for a user named Rich follows:

Schedule for Thu, Nov 18:
8:00 - 9:00   CS420 Operating Systems (teaching)
10:30 - 11:45 CS960 Theory of Database
12:00 - 1:00  Lunch
3:00 - 3:30   virg rich beth rod
Confirmed by:  virg rich beth
               To Discuss Curriculum Changes.
               Meet in F112.

4:30 - 5:00   R1 virg rich
               to discuss UNIX-OS/32 Networking.

(I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it->
Some discussion of the displayed sample schedule is warranted. The first three appointments are personal appointments that generally only apply to Rich's schedule. The 3:00 - 3:30 meeting is with four members, including Rich. Members Virg, Rich, and Beth are confirmed members; that is, they have all affirmatively answered a request for the meeting. Rod has not answered the request either way, from the fact that his name is in the "members" set, but not in the "confirmed" set. The "confirmed by" field in an indication that the meeting was originally requested by Rich, so that he is the owner of that meeting. The "confirmed by" field does not appear for that meeting on the other member's schedules for that day. 5:00 appointment shows up as a request, indicated by the "R" field. This means that Rich still needs to answer the request for that appointment.

A discussion of making and answering requests is left to the next section of this chapter. They are briefly mentioned here, as the entries which contain members are special cases for the delete routines (to be discussed).

The choices of input can now be dealt with. The first possibility calls for an insert process to be invoked. This process logically requires two pieces of information from the user, the time span of the appointment and a description of the appointment. Note that all appointments involving more that one user (referred to as "meetings") are not
inserted here, but through the separate request and answer procedure. After an 'I' is entered as an answer to the last prompt, the time span for the appointment to be inserted is asked for:

Enter Beginning and Ending Times. ->

after which the times can be entered. The formats in which this information is entered can vary. For instance, colons ("[:]"") may be either included or omitted, as can "AM" or "PM". In the latter case, certain assumptions are made. In particular, 1:00 - 6:55 times are assumed "PM", and 7:00 - 12:55 times are assumed "AM". Thus, in order to schedule an appointment at a time outside of these defaults, the "AM" or "PM" must be included, such as for 7:30 PM, or 6:00 AM.

The electronic calendar system will not allow conflicts. If the time entered in any way conflicts with any previous appointment of that day, the user is informed of the conflict and returned to the previous prompt. After a non-conflicting time span is entered, a prompt for the appointment description is displayed. An example of an insertion of Rich's schedule follows, as a continuation of the previously displayed sequence. The first insert attempt is a conflict.

(I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it -> I
Enter Beginning and Ending Times. -> 11:30 12:00
11:30 - 12:00 conflicts with a previous
10:30 - 11:45 appointment.
(I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it -> I
Enter Beginning and Ending Times. -> 11:45 12:00
Enter Description of appointment.
At this point a description can be entered. The description can be up to 80 characters, and is terminated by a carriage return.

The insert process just described can also be applied the same way to a general weekday's schedule. The only difference is that the insertion applies to more than just the one day's schedule. A general weekday schedule is indicated by the heading. For each insertion the entered time span is the compared with to flag conflicts. Upon completion of entering the description, an attempt is made to insert the new appointment to all days present in the calendar system of the same weekday. The results of the attempt are then displayed.

To illustrate, if the above insertion was made to a general Thursday schedule instead of the specific Thursday, November 18 schedule, the response after the appointment description was entered might have been:

11:45 - 12:00 appointment inserted to Thu, Nov 18.
11:45 - 12:00 appointment inserted to Thu, Nov 25.
Cannot insert to Thu, Dec 1, due to previous
11:30 - 12:00 appointment.
Insertion made to general Thu schedule.

If the appointment is desired for December 1, it would have to be manually inserted, after a deletion of the conflicting appointment of the same day. The dates for which the trace message are displayed are limited to those dates which have appointments scheduled other than weekly appointments. The change made implicitly applies to all succeeding Thursdays.
At this point the insertion process is concluded, and
the higher level prompt is displayed:

(I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it ->

The next possibility for an answer to this prompt is
delete, which behaves similarly to insert, but is even
simpler to use. All that is required by the delete process
is to know which appointment to delete. The easiest way to
uniquely identify an appointment is by its beginning times.
The prompt is:

Enter beginning time of appointment to delete. ->

The expected answer to this prompt should follow the same
format and defaults at the prompt for beginning and ending
times described earlier for the insert process, except, of
course, that only one time need be entered. The case where
the time entered does not exist as the beginning time of an
appointment is indicated by a prompt, such as:

Appointment with beginning time 11:45 not found.
(I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it ->

and control is then returned to the previous prompt.

From the user's point of view, the one entry for a
beginning time is all that is required. But the effects
resulting from special cases of deletes need to be defined.

If the schedule from which the appointment was deleted
was a weekday schedule, then the delete is also attempted on
all successive dates having the same weekday. Similar trace
messages are displayed. As an example:
11:45 - 12:00 appointment deleted from Thu, Nov 18.
11:45 - 12:00 appointment deleted from Thu, Nov 25.
11:45 - 12:00 appointment not found on Thu, Dec 1.
11:45 - 12:00 appointment deleted from Thu schedule.

Another case of delete is when the appointment to be deleted is a meeting of several users; that is, the appointment entry contains a list of participants. If such a meeting is deleted, then one of two events occur. If the user performing the deletion is the owner of the meeting; that is, the one that requested the meeting in the first place, then the meeting is considered cancelled. It is automatically deleted from all of the member's schedules, and mail is sent to each member informing him/her of the cancellation. If, instead, the user performing the deletion is a member of the meeting, but not the owner, then the deletion is only made to his/her schedule, but his name is automatically removed from the list of members on all of the member's schedules. Also, mail is sent to the owner of the meeting informing him/her of the omission of the one member from the meeting.

Three other possibilities for answers to the second level prompt exist, other than insert or delete. The "List" and "Newday" options perform backtracking. The "List" re-displays the day's schedule and the same prompt, and the "Newday" option displays the prompt for date selection (discussed previously), so that a different date/day/day can be observed or edited. The "Exit" choice returns control to the initial main menu, at the next higher level.
3.2 Meeting Requests

The second choice available at the main menu level is the "Make or Answer Requests" selection. The processes included in this subdivision are so categorized from the fact that all the calendar entries dealt with by this division are meetings, all including more than one member and an "owner" of the meeting. As previously described in the requirements / specifications, a meeting is first requested by one user to several members. When the requests are all answered affirmatively, the meeting in considered confirmed.

The first prompt after selection of this subdivision from the main menu is:

   (R)equest a meeting, or (A)nswer requests?  ->
If an "R" is entered, the request process is invoked. This process is similar to the insertion process previously described, in that a date, time, and description must be entered by the user, and that the appointment is inserted to the user's schedule for the particular data. The set of prompts making up the insertion process is augmented with additions allowing the insertion to apply to several members, as requests. The next prompt observed for the request process is:

   With whom would you like an appointment?
Enter names or list.  ->

Here the names entered must match the login names known by the system under which the electronic calendar is
implemented. The "list" option is often useful when the names are not absolutely known; the names of all users participating in the electronic calendar system (as they must be entered) will be listed, and the prompt will be re-displayed. An example with four total users is:

With whom would you like an appointment?
Enter names or list. -> list
  beth virg rich rod
With whom would you like an appointment?
Enter names or list. -> beth virg
For how long? ->

The value entered for the duration of the proposed meeting must be a multiple of 5 (5, 10, 15, ...). The next prompt asks for the date or month and date of the proposed meeting:

Enter Preferred date, or month and date. ->

The response to this prompt follows the same guidelines as the similar prompt at the beginning of the "Update or Observe" process; a null line implies the current date, a single date can be entered for dates up to two weeks from the current date, and past that date the month name is required. The only exception is that weekday names are not allowed. The next prompt is:

Standard 8:00 - 5:00? (Y/N) ->

A "yes" answer to this prompt will limit meeting possibilities to the time spans between 8:00 AM and 5:00 PM. Otherwise, all of the possibilities during the 24 hour day will be listed.

At the successful completion of a date selection and the decision of the "Standard?" question, the "free time
intersection" of all the members entered and the user are calculated and displayed. An example of the display is:

Here are the free times for Thu, Nov 18:

9:00 - 10:00       2:00 - 2:30

Do you want one of these? Y/N/e(X)it

The time slots listed represent the free time intersection of all the members (including the owner) for that particular day. Only the time pairs that represent durations greater than or equal to the previously entered time span are displayed.

The associated prompt asks for one of three choices. A "yes" answer to the prompt means that the user does wish to proceed with the requesting process, on the date to which the displayed possibilities apply. In that case the prompt for beginning and ending times of the requested meeting appears:

Enter Beginning and Ending Times. ->

The format for the expected input is exactly the same as the time pairs expected by the insertion process, described in the previous section. The time pair entered is similarly validated by checking against the formerly displayed times for conflicts. After a conflict - free time pair is entered, a prompt for the meeting description is displayed, again analogous to the insertion process:

Enter description of meeting. ->

When the description is entered the request process is
complete. The meeting will appear on all the member's schedules with the "R" field present. In addition, mail is automatically sent to the members, informing each of the request, with the date included, so that their acknowledgement (answer) can be as prompt as possible.

For the prompt:

Do you want one of these? Y/N/e(X)it ->

there are two alternatives for answers other than "yes". A "no" response backtracks to the date selection prompt, so that the request can be attempted on an alternate day. The "exit" choice returns control to the main menu, without a request being made, or mail being sent.

To perform the opposite function, of answering requests made by other members, the second choice to the prompt:

(R)equest a meeting, or (A)nswer requests? ->

is selected. At this point the user has been informed that another calendar user requests his/her presence at a meeting via system mail. The mail contained the owner of the meeting (the sender of the mail), and the month and date of the proposed meeting.

The first prompt observed, upon invocation of the "answer" choice, is:

Enter date for which requests are to be answered. ->

The input expected here is the same as that expected in the previous date selection prompt of the "request" process. A date without the month name can be entered for dates up to
two weeks from the present day, and past that date the month
name is required.

If the date selected does not contain any entries in
which the user has requests, a prompt such as:

No requests exist on Nov18.
appears, and control is returned to the main menu. If,
instead, a request does exist for the date selected, then
the schedule is displayed, in the same format as described
in the "Update or Observe" section of this chapter. Another
example follows, again for a user named Rich:

Schedule for Tue, Nov 16, 1982:
8:00 - 9:00    Unavailable
1:00 - 2:00    R1 virgil rich
                For spring scheduling possibilities.
2:00 - 3:00    R2 beth rich
                to discuss an alternative UNIX
distributed database system.

R1?  (A)cept, (R)equest, or (M)ove-on ->
For each request, the user will be given this prompt. In
this case after the first one is answered, the same prompt
for the second request (the one with Beth) will be
displayed, as:

R2?  (A)cept, (R)equest, or (M)ove-on ->
For each prompt, the results of the answers are these. If
the user accepts the request, then the "R" field for that
meeting is removed from his schedule for that day, the
meeting is no longer considered a request. Also, mail is
sent to the owner of the meeting informing him/her that the
user here has decided to accept the request for a meeting. If, instead, the user decided to reject the request for a meeting, his/her name is removed from the "members" field of all the other members of the meeting. Mail to the owner informs him/her that the user has rejected the meeting request. The meeting is then deleted from the users schedule.

As a final possibility, the "move-on" choice may be taken. This choice is designed for use when the decision to accept or reject a meeting request cannot be made at the present time. The request remains on the schedule unaltered, and no mail is sent.
CHAPTER 4
FUTURE ENHANCEMENTS

As with any software product, this implementation has limitations and a finite scope. While the design is consistent with the stated specifications, future changes in the demands for the service provided by this electronic calendar system could necessitate accommodating changes to software. The enhancements described here are, in the writer's opinion, the most probable cases. There is, of course, an unlimited number of possible future demands for this and any other ongoing software service.

The six month look-ahead limit is a somewhat arbitrary requirement. The particular application in which this implementation is to be used, primarily the KSU Computer Science department, does not require appointments of any sort to be made past that time interval. However, if conditions were ever to change, such that this assumption was no longer true, the software change could easily be made. The change would require a minimal increase in computer memory.

There exists a possibility in some applications that a (possibly varying) inter-appointment time slot would be implemented, a case where it would not be desirable to allow any appointment to begin at the same time the previous appointment ended. This is not implemented. The
justification is, again, that the application does not necessitate such a feature.

Access security is not currently explicitly provided by the software, but is implicit in the UNIX operating system. An authorized user is one that has possession of a valid account on the UNIX system. Account ownership entitles a user to maintain a personal calendar, and to make requests for meetings with other users. Other electronic calendar applications might require a "screening" process to assign security rank to each user, with access privileges dependent on the rank. The changes to software to implement this feature would be substantial, but conceivable.

"Tickler" files, or automatic reminders, could be integrated in an electronic calendar system. They would, at the user's option, provide automatic reminders of any desired appointments at a specified time. This is not implemented for two reasons. First, it would require a high degree of operating system interaction, creating a detriment to portability. The second reason is that such a feature already exists under the UNIX operating system, for which this project will apply, that adequately provides this service.

With an increase in use of the electronic calendar system and/or expansion of facilities to distributed systems, a distributed electronic calendar system could become desirable. Conversion of the present implementation
to one that includes inter-system communication is conceivable, since the source code is written in the fairly standard version of the Pascal language. The conversion would entail a re-Compilation (or re-interpretation) for the new system, and the means for the programs to read and write to files across network lines.
REFERENCES


APPENDIX 1
USER'S QUICK REFERENCE

Contained in this appendix is an outline-like representation of the sequence of prompts and expected user responses for the writer's implementation of an electronic calendar system. The intention is that this section be used as a quick reference or general guide for a user unfamiliar with the formats of input expected by the software.

The representation of prompts and responses is divided into two categories. These categories correlate with the two choices available to the user at invocation of the electronic calendar. The display is:

Welcome to Electronic Calendar.
Today is Thu, Nov 18, 1982.
Main Menu:
   1. Update or Observe Personal Calendar.
   2. Make or Answer Meeting Requests.
   X. Exit.
Please Enter Command Number or X. ->
{1, 2, X, x}

The choices for input will be represented in set notation. In this case, the prompts and responses following choice "1" are described on the following page, choice "2" is described on the succeeding page. Labels (to which GO TO statements refer) are shown in asterisks (eg. *A**). Text in pointed brackets (<..>) is not actually displayed or required, but describes in english either an instruction, or the actual data element that will appear or be expected.
Update or Observe Personal Calendar

***A*** <ret> for today's schedule, enter date or day. ->
{<ret>, 18, Nov18, nov18, Wed, wed, ... }

***B*** <The schedule for the date entered is displayed>

***C*** (I)nsert, (D)elete, (L)ist, (N)ewday, or e(X)it? ->

case

{I, i}: Enter beginning and ending times. ->
{800 900, 8:00 AM - 9:00 AM, ... }

Enter description of appointment.
{80 characters or less}

Insertion Made
<GO TO ***C***>

{D, d}: Enter beginning time to delete.
{800, 8:00, 8:00 AM, ... }

Deletion Made.
<GO TO ***C***>

{L, l}: <GO TO ***B***>

{N, n}: <GO TO ***A***>

{X, x}: <exit to main menu>

dose case.
Make or Answer Meeting Requests

(R)equest a Meeting, or (A)nswer Requests? ->

case

{R, r}: With whom would you like an appointment?
   Enter names or list. ->
   {list, virg, virg beth, ...}

For how long? (minutes) ->
   {5, 10, 15, 20, ...}

Standard 8:00 - 5:00? (Y/N) ->
   {Y, y, N, n}

***D***

Enter Preferred date. ->
   {18, Nov18, nov18, ...}

Here are the possibilities for
Thu, Nov 18:
<time pairs representing free times.>
Do you want one of these? Y/N/e(X)it ->

case

{Y, y}: Enter beginning
   and ending times.
   {800 900, ...}
   Enter meeting description.
   {<80 characters>}
   <Exit to Main Menu>

{N, n}: <GO TO ***D***>

{X, X}: <Exit to Main Menu>

dcase.

{A, a}: Enter date for which requests are to
   be answered. ->
   {18, Nov18, nov18, ...}

<display of schedule, requests
   numbered as R1, R2, ... Rn>

R1? (A)ccept, (R)eject, (M)ove-on? ->
   {A, a, R, r, M, m}

dcase.
APPENDIX 2

DESIGN / MAINTENANCE MANUAL

2.0 Introduction

This appendix contains a brief summary of the design of the writer's implementation of an electronic calendar system. The design used and discussed here represents one possible solution to the problem of meeting the specifications called for in the problem statement of chapter two in this report.

The rationale for this section is two-fold. A design document of a specific implementation can often give insight to a (possibly improved) design solution for this or any similar application. A specific solution to any problem is the logical "final chapter" in a complete problem description. The second use for this appendix is as a maintenance manual for use by the operator(s) of the UNIX system, since some understanding of the design is needed by those who maintain it. Certain "housekeeping" chores related to file management and access rights must be initiated by the systems operator in order for the software to be usable. Also, future changes to existing software necessitate a basic knowledge of the workings of the programs. The first section is a general statement of design with respect to the software's interaction with its external environment. The facets dealing with lower-level design issues are omitted, and left to internal code
documentation. The next section addresses the specific maintenance problem of adding a new user to the electronic calendar system, and the last section looks at a proposed archival file system to "backup" schedules of past dates.

2.1 Design

The electronic calendar system relies heavily on the UNIX operating system for storage and access of external files. A differentiation between short-term and long-term dates is noted by the user when he/she specifies a date to observe. It was stated that a short term date (one less than 14 days away) could be accessed by entering the date only. Future dates which are more that 14 days from the present require the month name and the date. Except for these details, the user has been hidden from the distinction, but in implementation the differences are more acute.

At any point in time, the software requires that the user has contained in his/her UNIX account a directory (presently called "caldir") containing 14 files representing the 14 short-term days. These files are named by the character pair conversion of the date which the file represents. So the filename for date 1 is "01", 10 is "10", etc.

Also required are seven files that represent the schedules of the seven weekday that apply globally to all
dates. They are named by the capitalized three-letter abbreviation of the weekdays ("Mon", ..., "Sun").

Files representing long-term dates (greater than or equal to 14 days from the present) are also stored in individual files. They are named by the concatenation of the month and date (e.g. "Nov18", "Dec03"). Since the number of long-term dates present in a user's directory varies (in a way to be discussed), a separate directory file, called "direct" contains a listing of the long-term files in existence.

To summarize the required files in a user's directory, the numbers of each type are: (14) short-term files named by dates, (7) weekday files named by each weekday, (1) directory file named "direct", which contains N records of long-term filenames, (N) long-term files, named by month/date.

The task of maintaining the above mentioned files in non-trivial, but fortunately it is handled completely automatically by two Pascal programs that make up the electronic calendar system. The first program is invoked at random by the user, to perform any of the desired tasks described in chapter three. The second is a program whose execution is invoked automatically by the operating system. This invocation is daily at 11:00 PM. The effects of these programs on the status of the files can now be explained.

As a user is running the main Pascal program, access is
provided to any date's schedule up to a year from the present day. But 365 files are not kept on the file space of his/her account. As a general rule or convention, the following statement can be made: "Only long-term dates in which the schedules on those dates differ from the corresponding weekday's schedules are kept on file." In other words, if a long-term date is a duplicate of the corresponding weekday of that date, then there is no need to have the file duplicated. The following example which traces the main program's execution elucidates further.

Suppose a user indicates a desire to observe or edit the schedule for June 10, 1983, a long-term date. The file "Jun10" might or might not exist in the user's file space. To display the schedule for that date, the file "direct" is searched for the record containing "Jun10". If the record is found then the file with same name is read, and the resulting schedule displayed. If instead, "Jun10" is not found in the file "direct", then a procedure is called to determine which day of the week June 10, 1983 falls on. The file corresponding to that weekday is read, and the resulting schedule is displayed. The heading for the display is still "Schedule for Fri, Jun 10, 1983:", but, appropriately, the user doesn't know (or care) if the information came from the file "Jun10" or "Fri".

If an edit (insertion or deletion) is then made to the June 10 schedule and the file did not previously exist, then
a new file "Jun10" is created and the edited schedule written to it. Also, a record containing the "Jun10" information is added to the directory file, implying the existence of a new external file.

A somewhat reversed process is performed by the system invoked program. Its task is to create a new file whose name will be included in the following day's set of short-term dates. Two cases analogous to the ones described above are handled. After determination of the day, month, and date of the new date, the directory file is searched for the corresponding record. If it is found, the file representing the long-term date is copied to to short-term date's file, the record is removed from the directory, and the long-term file is removed. If instead, the long-term date is not found in the directory, the corresponding weekday is copied to the new short-term day's file. Again an example will help to clarify.

Suppose that the automatic program is executed on Monday, November 1, 1982 (at 11:00 PM). The first step is to determine information about the new short-term day (from tomorrow's point of view). The results would yield the fact that Monday, November 15, 1982 will be considered short-term tomorrow, so the file "15" must be created before then. The directory is searched for a record containing "Nov15". If it is found, then (1) file "Nov15" is copied to file "15", (2) the record containing "Nov15" is removed from the
directory, and (3) file "Nov15" is removed. If, instead, a record containing "Nov15" is not found in the directory, the file "Mon" is copies to file "15".

Another issue that will be addressed is the design description is of how the software communicates with the operating system. With most implementations of a language the language is augmented with system - specific calls to the operating system. At the time of writing, no such facility exists in the interpreted Berkeley Pascal. To implement an interface to needed system functions, the programs write to an executable text file with user level commands. At the termination of the Pascal program's execution, the text file (named "execute") is submitted to the "UNIX" system "shell" for execution.

The possibilities for commands contained in "execute" are as follows. In the main user program, execution commands are written to the executable file, so that mail will be sent upon program termination. Such mail occurs when making or answering meeting requests, or when deleting a meeting from a date's schedule where the deletion affects other members. In the automatic program, "execute" is written to commands to copy ("cp"), and possibly remove ("rm") files, depending on the circumstances.

2.2 Adding a user to the electronic calendar system.

The most commonly performed maintenance task will be
the addition of a new user to the electronic calendar system. The processes involved in doing this entail two distinctly different tasks; creating the initial calendar data files, and editing the source code of the main program, so that it will recognize the addition of a user, and can be called for execution by that user.

The required data files that will reside in each user's "caldir" directory have been defined in the previous section. These files must be declared, or allocated for the calendar system, even though they will be initially empty. The files include 14 date files, 7 weekday files, and one file named "direct".

The source code of the main Pascal program requires two changes. The first can be found near the very beginning of the global declarations. The new user's name should be added to the list of enumerations making up the type "users". The first element of that type with a "notused" name can be replaced by the users' "logon" name. The second change occurs at the first procedure, "InitGlobalVars". An assignment statement assigning an element of the array "CvtName" (indexed by the user's name) the value of the user's name in string form should be added. Also, the user's name enumerated is the new "LastMember" be added. Also, the enumeration element corresponding to the user's name is assigned as the new value of "LastMember". Thus, that assignment statement should be changed.
1.3 Backups / Archival Files

At the time of writing, no means for an archival file system is implemented. Past dates are currently inaccessible by the user. If an archival file system becomes desirable, program changes would reflect the current availability of computer memory.

One such possibility would involve a change to the automatic program, so that in addition to creating a new file representing the new short-term date, the schedule for the "soon-to-be-lost" date would be appended to an archival file. The archival file would periodically be dumped to magnetic tape, and then erased from disk.

If a text file is to be used for archival, then the "WriteArray" routine should be extracted from the main user program, and called by "GetNewDay" of the automatic program. The call will include information (as parameters) about the current day. Since Pascal doesn't contain facilities to append text to files, the program will have to solve the problem by writing UNIX "rename" and "concatenate" commands to the external "execute" file.
APPENDIX 3

PASCAL SOURCE CODE
program ecal (input, output, f, direct, mailer, execute);

const
  MaxFileLength = 40;
  MaxDirLength = 50;

type
  FakeBoolean = (yes, no);
  FileLengthRange = 0..MaxFileLength;
  users = [david, clq, beth, rich,
           virg, notused1, notused2,
           list];
  UserSet = set of users;
  AppointmentTypes = (request, weekly, other);
  TimeString = packed array [1..8] of char;  {eg. '10:00 AM'}
  string = packed array [1..80] of char;
  rec = record
    b, e : integer;  {beginning and ending times}
    t : AppointmentTypes;  {request or other}
    s : string;  {80 character description}
    case setexists : boolean of
      true : ( owner : users;  {requester}
              use : UserSet;  {members of meeting}
              confirmed : UserSet );  {confirmed members}
    false : ()
  end;
RecFile = file of rec;  {calendar files, one per day}
RecArray = array [FileLengthRange] of rec;

pair = packed array [1..2] of char;  {assorted strings}
triple = packed array [1..3] of char;
string5 = packed array [1..5] of char;
string8 = packed array [1..8] of char;
string30 = packed array [1..30] of char;
DaysOfMonth = 0..31;
DayDescrip = record
  datechar : pair;
  day, month : triple;
  year : integer
end;
DaysArray = array [DaysOfMonth] of DayDescrip;
DaysSet = set of DaysOfMonth;  {will contain 14 short-term dates}

TimePair = record
  BeginTime,
  EndTime : integer
end;
TimePairArray = array [FileLengthRange] of TimePair;

DirRec = record
  Day : triple;
  MonthDate : string5
end;
DirRecFile = file of DirRec;
DirLengthRange = 0..MaxDirLength;
DirRecArray = array [DirLengthRange] of DirRec;
MailRec = record
    WrittenTo : FakeBoolean;
    message : text
end;

MailRecArray = array [users] of MailRec;

var
    ThisUser : users;
    dset : DaysSet;
    dar : DaysArray;
    today : DaysOfMonth;
    CvtName : array [users] of string8;
    FirstMember, LastMember : users;
    code : char;
    f : RecFile;
    direct : DirRecFile;
    mailer : text;
    execute : text;
    MailAr : MailRecArray;
{***** Init Global Vars *****}
{
Called by
main program

External
argv    : Berkeley Pascal built in. Accepts the program parameter, which
         is the user's name.
CvtName  : String array to convert enumerated names to strings.
MailAr   : Array of local text files to which mail messages are stored.
FirstMember,
LastMember : Pseudo-Constants representing the first and last members
            of the enumerated names of users.
}

procedure InitGlobalVars;
var
  ThisMember : users;
a : string8;
begin
  FirstMember := david;
  LastMember  := virg;
  CvtName [david] := 'david';
  CvtName [clq]   := 'clq';
  CvtName [beth]  := 'beth';
  CvtName [rich]  := 'rich';
  CvtName [virg]  := 'virg';
  for ThisMember := FirstMember to LastMember do
    with MailAr [ThisMember] do
      begin
        WrittenTo := no;
        rewrite (message)
      end;
  argv (1,a);
  ThisUser := FirstMember;
  while (ThisUser < LastMember) and (CvtName[ThisUser] <> a) do
    ThisUser := succ(ThisUser);
  if CvtName[ThisUser] <> a then halt
begin
{**** Dump Local Mail Files ****}
{
For each of the local mail files that have been written to during the
course of program execution, the contents of the local file is copied
to an external file, one given the name of the destination of the mail.
The external file is the 'letter' which will be sent at program termination.
Called by
main program
External
   FirstMember,
   LastMember : Endpoints (bounds) for the set of users.
   MailAr : Array of local text files.
   mailer : General name for external text file containing message.
}
procedure DumpLocalMailFiles;
var
   ThisMember : users;
   c : char;
begin
   for ThisMember := FirstMember to LastMember do
   with MailAr [ThisMember] do
     if WrittenTo = yes then
       begin
         writeln (execute,'mail ',CvtName(ThisMember),' < ',CvtName(ThisMember));
         rewrite (mailer, CvtName(ThisMember));
         reset (message);
         while not eof (message) do
           begin
             while not eoln (message) do
               begin
                 read (message,c);
                 write (mailer,c)
               end;
             readln (message);
             writeln (mailer)
           end
         end;
       end;
   end;
{##### tme, scale #####}
{
For easier time calculations and comparisons, the 24 hour clock is mapped onto
a 288 element scale, where each element represents a five minute interval.
'tme' converts a scaled value to a readable time, and 'scale' converts a
numeric 24 hour time to a scaled value. The mapping is:
1:00 AM = 0, 1:05 AM = 1,...,8:00 AM = 84,...,1:00 PM = 144,...
Called by
   All procedures dealing with times and times I/O.
}
function tme (n:integer):TimeString;
var
  h,m,h1,h2,m1,m2:integer;
  temp:TimeString;
begin
  if n >= 144 then
    begin
      n := n - 144;
      temp[7] := 'P'
    end
  else
    begin
      temp[8] := 'M';
      h := (n div 12) mod 12 + 1;
      m := (n * 5) mod 60;
      h1 := h div 10;
      h2 := h mod 10;
      m1 := m div 10;
      m2 := m mod 10;
      if h1=1 then temp[1] := '1'
        else temp[1] := '0';
      temp[2] := chr(h2 + ord('0'));
      temp[3] := '':'
      temp[4] := chr(m1 + ord('0'));
      temp[5] := chr(m2 + ord('0'));
      tme := temp
    end;
}
function scale (n:integer):integer;
begin
  scale := (n mod 100) div 5 + (n div 100 - 1) * 12
end;
{**** MthNum, Day Of Week, Dates Info ****} 

To determine which dates of the month are to be considered short-term dates, these procedures calculate and return the 14 short-term dates. Information about each day (day-of-week, month, and year) is also stored for easy access during program execution.

DatesInfo called by main program.

MthNum called by DatesInfo.
CheckExistence: for month-value comparison (eg. 'Jan' < 'Feb').

DayOfWeek called by DatesInfo.

External date: Berkeley Pascal built-in procedure that returns current day info.

Output parameters
dset: the set of 1..31 numeric values representing short-term dates.
dar: array indexed by members of dset, stores information about the short term dates.
today: numeric pseudo-constant, the numeric value of today's date.

} 

function MthNum (month : triple) : integer;
begin
    if month = 'Jan' then MthNum := 1
    else if month = 'Feb' then MthNum := 2
    else if month = 'Mar' then MthNum := 3
    else if month = 'Apr' then MthNum := 4
    else if month = 'May' then MthNum := 5
    else if month = 'Jun' then MthNum := 6
    else if month = 'Jul' then MthNum := 7
    else if month = 'Aug' then MthNum := 8
    else if month = 'Sep' then MthNum := 9
    else if month = 'Oct' then MthNum := 10
    else if month = 'Nov' then MthNum := 11
    else if month = 'Dec' then MthNum := 12
    else halt
end;

function DayOfWeek (date: DaysOfMonth; month: triple; year: integer): triple;
var
    DayNum: 0..6;
    MonthNum: 1..12;
    funct: array [1..12] of integer;
begin [function]
    MonthNum := MthNum (month);
    if ((month='Jan') or (month='Feb')) and (year mod 4 = 0) then 
        DayNum := (funct[MonthNum] + date + year + year div 4 - 1) mod 7
    else 
        DayNum := (funct[MonthNum] + date + year + year div 4) mod 7;
    case DayNum of
        0: DayOfWeek := 'Sat';
        1: DayOfWeek := 'Sun';
procedure increment(var this: triple);
begin
  if this = 'Sat' then this := 'Sun'
  else if this = 'Sun' then this := 'Mon'
  else if this = 'Mon' then this := 'Tue'
  else if this = 'Tue' then this := 'Wed'
  else if this = 'Wed' then this := 'Thu'
  else if this = 'Thu' then this := 'Fri'
  else if this = 'Fri' then this := 'Sat'
  else if this = 'Jan' then this := 'Feb'
  else if this = 'Feb' then this := 'Mar'
  else if this = 'Mar' then this := 'Apr'
  else if this = 'Apr' then this := 'May'
  else if this = 'May' then this := 'Jun'
  else if this = 'Jun' then this := 'Jul'
  else if this = 'Jul' then this := 'Aug'
  else if this = 'Aug' then this := 'Sep'
  else if this = 'Sep' then this := 'Oct'
  else if this = 'Oct' then this := 'Nov'
  else if this = 'Nov' then this := 'Dec'
  else if this = 'Dec' then this := 'Jan'
  else halt
end;

procedure DatesInfo(var dset: DaysSet; var dar: DaysArray; var today: DaysOfMonth);
var
  a : alfa;
  ThisDay, ThisMonth : triple;
  i, ThisYear, temp : integer;
  ThisDate, ThisMonthLength : DaysOfMonth;
begin
  date(a);
  if a[1] = ' ' then a[1] := '0';
  ThisDate := 0;
  for i := 1 to 2 do
    ThisDate := ThisDate*10 + ord(a[i]) - ord('0');
  today := ThisDate;
  for i := 4 to 6 do
    ThisMonth[i-3] := a[i];
  ThisYear := 0;
  for i := 8 to 9 do
    ThisYear := ThisYear*10 + (ord(a[i]) - ord('0'));
  ThisDay := DayOfWeek (ThisDate, ThisMonth, ThisYear);
  if (ThisMonth = 'Apr') or (ThisMonth = 'Jun') or
    (ThisMonth = 'Sep') or (ThisMonth = 'Nov') then
    ThisMonthLength := 30
  else if (ThisMonth = 'Feb') and (ThisYear mod 4 = 0) then
    ThisMonthLength := 29
else if (ThisMonth = 'Feb') then
  ThisMonthLength := 28
else
  ThisMonthLength := 31;
dset := [ ];
for i := 1 to 14 do
begin
  dset := dset + [ ThisDate ];
  dar[ ThisDate ].datechar[1] := chr( ThisDate div 10 + ord('0') );
  dar[ ThisDate ].datechar[2] := chr( ThisDate mod 10 + ord('0') );
  dar[ ThisDate ].day := ThisDay;
  dar[ ThisDate ].month := ThisMonth;
  dar[ ThisDate ].year := ThisYear;
  temp := ThisDate;
  temp := succ ( temp );
  increment ( ThisDay );
  if temp > ThisMonthLength then
  begin
    increment ( ThisMonth );
    if ThisMonth = 'Jan' then
      ThisYear := succ ( ThisYear );
    temp := 1
  end;
  ThisDate := temp
end
{**** Read String ****}
{
  For standard terminal input of packed character arrays (strings).
  Called by
    Insert : procedure to add an appointment. Reads appointment description.
    Request : procedure to make a meeting request. Reads meeting description.
  Output parameter
    s : 80 character array, from standard input. Text is delimited by '\$'.
}
procedure ReadString (var s:string);
var
  i : integer;
  first : boolean;
begin
  first := true;
  i := 1;
  while (not eoln) and (i<80) do begin
    read (s[i]);
    if (not first) or (s[i] <> ' ') then
      i := succ(i);
    first := false
  end;
  s[i] := '\$'
end;
{***** Write F String *****}
[
Write formatted string displays the 'string' field of an appointment or
meeting during terminal display of per-day calendars. The string is
displayed on the right half of the screen, if the string is too long
it is broken in half (between words) and displayed on two lines.
Called by
  WriteArray : procedure to display a day's schedule to the terminal.
Input parameters
    nl  : New line. If nl=true, then the string is displayed on
         a new line, and after tabbing. This would be required
         if the 'members' set was previously displayed on the
         first line of the meeting entry.
    s   : The 80 character array to be displayed.
]
procedure WriteFString (nl:boolean; s:string);
var
  i : integer;
begin
  if nl then write (' ':26);
  i := 1;
  while (s[i]<>') and ((i<40) or (s[i]<>')')) do begin
    write (s[i]);
    i := succ(i)
  end;
  if s[i]>'#' then begin
    i := succ(i);
    writeln;
    write (' ':26)
  end;
  while s[i]>'#' do begin
    write (s[i]);
    i := succ(i)
  end
end;
[**** Read Single Time ****]
{
A single time-of-day is read and mapped onto a 24 hour clock. Assumptions are made in the case of a missing 'AM' or 'PM'.
Called by
    delete : As the beginning time of the calendar entry to delete.
Output parameter
    n : Numeric value of time on a 24 hour clock (eg. 2:00 PM = 1400)
}
procedure ReadSingleTime (var n:integer);
var
  digits : set of char;
  ok : boolean;
  c : char;
begin
  digits := ['0'..'9'];
  repeat
    ok := true;
    n := 0;
    repeat
      read (c);
      if c in digits then n := n*10 + ord(c) - ord('0')
    until (eoln) or (not (c in digits + [':',',' ']'));
    while (not eoln) and (c = ' ') do read (c);
    if n mod 5 > 0 then
      begin
        ok := false;
        writeln ('Time should be a multiple of 5.')
      end
    else if (n div 100 < 1) or (n div 100 > 12) or (n mod 100 > 55) then
      begin
        ok := false;
        writeln ('Time ',n div 100:2,':',n mod 100:2,' is not legal.')
      end;
    if (c = 'P') or (c = 'p') or ((c <> 'A') and (c <> 'a') and (n < 700))
      then n := n + 1200;
    if not ok then
      begin
        readln;
        write ('Please reenter time. -> ')
      end
    until ok
end;
{***** Read Time Pair *****}
{
Analogous to 'ReadSingleTime' (above), except two values are read in.
Called by
insert : as the beginning and ending times for an appointment insertion.
request : as the beginning and ending times for a meeting request.
}
procedure ReadTimePair (var b,e:integer);
var
digits : set of char;
ok : boolean;
c : char;
begin
digits := ['0'..'9'];
repeat
  ok := true;
b := 0;
  repeat
    read (c)
  until c <> ' ';
  while c in digits + ['::'] do
begin
  if c <> ':' then
    b := b*10 + ord (c) - ord ('0');
  read (c)
end;
if b mod 5 > 0 then
begin
  ok := false;
  writeln ('Beginning time should be a multiple of 5.')
end
else if (b div 100 < 1) or (b div 100 > 12) or (b mod 100 > 55) then
begin
  ok := false;
  writeln ('Beginning time ','b div 100:2','::','b mod 100:2,' 'is not legal.')
end;
while c = ' ' do read(c);
if (c = 'p') or (c = 'P') or ((c <> 'a') and (c <> 'A')) and (b < 700) then
begin
  b := b + 1200;
end;
while not (c in digits) do read (c);
e := ord (c) - ord ('0');
while (not eoln) and (c in digits + ['::']) do
begin
  read (c);
  if c in digits then e := e*10 + ord (c) - ord ('0')
end;
while (not eoln) and (c = ' ') do read (c);
if e mod 5 > 0 then
begin
  ok := false;
  writeln ('Ending time should be a multiple of 5.')
end
else if (e div 100 < 1) or (e div 100 > 12) or (e mod 100 > 55) then
begin
ok := false;
  writeln ('Ending time ', e div 100:2,':',e mod 100:2,' is not legal.')
end;
if (c = 'F') or (c = 'P') or ((c <> 'A') and (c <> 'a') and (e < 700))
  then e := e + 1200;
if e < b then
begin
  ok := false;
  writeln ('Time ',tme(scale(b)),' - ',tme(scale(e)),' is impossible.')
end;
if not ok then
begin
  readln;
  write ('Please reenter beginning and ending times. -> ')
end
until ok
end;
{**** Read Time Span ****}
{
Input of a numeric time span length. The number of minutes desired is input.
Called by
    Request: For the value of the minimum meeting length required by the requestor.
Output parameter
    n : The minutes. A multiple of 5.
}
procedure ReadTimeSpan (var n : integer);
var
    c : char;
    ok : boolean;
begin
repeat
    ok := true;
    n := 0;
repeat
    read (c);
    if c in ['0'..'9'] then
        n := n*10 + ord (c) - ord ('0')
    until (not (c in ['0'..'9'])) or eoln;
if n = 0 then
    begin
        ok := false;
        writeln (c, ' is not a valid time.')
    end
else if (n mod 5 <> 0) then
    begin
        ok := false;
        writeln ('Time span must be a multiple of 5.')
    end;
if not ok then
    begin
        readln;
        write ('Please reenter time span in minutes. -> ')
    end
else
    begin
        n := n div 5
    until ok
end;
**** Write Set ****
{
For terminal display of members of a meeting, confirmed members of a meeting.
Called by
    WriteArray : during display of a particular day's appointments, when a
    several member meeting is to be displayed.
Input parameter
    uset : the set of members (enumerations).
}
procedure WriteSet (uset : UserSet);
var
    user:users;
begin
    for user := FirstMember to LastMember do
        if user in uset then
            write (CvtName [user])
end;
{### Read Set ###}
{
Input of string names, conversions to enumerations, and addition to the set.
Called by
Request : When prompting for the list of members desired for the meeting.
Output parameter
s : The set of members (enumerated) for the requested meeting.
}
procedure ReadSet (var s : UserSet);
var
   this : string8;
   ThisMember : users;
   str : string;
   i, j : integer;
   ok : boolean;
begin
   repeat
      ok := true;
      s := [];
      ReadString(str);
      i := 1;
      while str[i] <> '#' do begin
         while (str[i]<>'#') and (str[i]=' ') do i := succ(i);
         if str[i]<>'#' then begin
            j := 1;
            this := ' ';
            while (str[i]<>'#') and (str[i]<> ' ') and (j<8) do begin
               this[j] := str[i];
               i := succ(i);
               j := succ(j)
            end;
            ThisMember := FirstMember;
            while (this<>CvtName(ThisMember)) and (ThisMember < LastMember) do
               ThisMember := succ (ThisMember);
            if this = CvtName(ThisMember) then
               s := s + [ThisMember]
            else if (this[1] in ['l','L']) then
               s := [list]
            else begin
               writeln (this,'is invalid. Please reenter names, or list. ->');
               readln;
               ok := false
            end
         end
      end
      until ok
   end;
}
{ ***** Read From File ***** }
{
Copies an external file to a similarly typed array for observation, or editing. Assumes that file 'f' has been previously reset to the desired file.
Called by
  UpdateOrObserve : for observation or manipulation.
  TemplateInsert : for reading all the files of a certain weekday.
  TemplateDelete : for reading all the files of a certain weekday.
  Request : for reading a schedule in which a request is to be made.
External
  f : file.
Output parameters
  ar : array of appointments / meetings.
  l : length of array.
}

procedure ReadFromFile (var ar:RecArray; var l:FileLengthRange);
begin
  l := 0;
  while not eof (f) do
  begin
    l := succ (l);
    read (f, ar[l])
  end
end;

{ ***** Write To File ***** }
{
Opposite of 'ReadFromFile' (above). Updates external file 'f' to the current value of the appointment / meeting array. Assumes 'f' has been rewritten to the desired external file.
Called by
  UpdateOrObserve : when changes have been made to a schedule.
  Answer : to carry out the changes of an 'accept' or a 'reject'.
External
  f : file.
Input parameters
  ar : array of appointments / meetings.
  l : length of array.
}

procedure WriteToFile (ar:RecArray; l:FileLengthRange);
var
  i : FileLengthRange;
begin
  for i := 1 to l do
  begin
    write(f, ar[i])
  end
end;
[**** Write Dir To File ****]
{
Analogous to 'WriteToFile' (above), except for use when writting a directory
of long term dates back to the external file 'direct'.
Called by
UpdateOrObserve: when a long-term date's schedule is to be added.
Request: If a request caused a new long-term date.
Input parameters
ar: array of long-term date's information.
l: length of array.
}
procedure WriteDirToFile (ar:DirRecArray; l:DirLengthRange);
var
  i: DirLengthRange;
begin
  for i := 1 to l do
  begin
    write (direct, ar[i])
  end
end;
{**** Write Array ****}
{
For terminal display of a single day's schedule.
Called by
  UpdateOrObserve : at input of a 'list' command by the user.
  Answer : so the user can see the the requests are that need
          to be answered.
External
  WriteSet : procedure to convert enumerated names of a set to
             character strings and output to terminal.
  WriteFString : procedure to display the 80 character description
                 of the appointment / meeting formatted on two lines, if
                 need be.
Input parameters
  ar : schedule to be displayed.
  l : length of array.
  weekday : to adjust heading of display for either a specific day
            or a general weekday (that applies to all weeks).
  day : day of the week.
  month : month, used for specific dates.
  date : numeric date, also for specific dates only.
}
procedure WriteArray (ar:RecArray; l:FileLengthRange; weekday:boolean;
                     day, month:triple; date, year:integer);

var
  i,r : FileLengthRange;
begin
  writeln; writeln; writeln;
  if weekday then
    writeln ('General Schedule for ',day,' : ')
  else
    writeln ('Schedule for ',day,' , ',month,date:3,' , 19','year:2,','');
  r := 1;
  if l = 0 then
    writeln (' (no appointments scheduled)');
  for i := 1 to l do
    with ar[i] do begin
      writeln;
      write (tme(b),' - ',tme(e));
      if (t = request) and (owner <> ThisUser) then begin
        write (' R',r:1,' ');
        r := succ (r)
      end
      else
        write (' ');
      if setexists then
        begin
          WriteSet (uset);
          writeln;
          if owner = ThisUser then
            begin
              write ('Currently confirmed by : ');
              WriteSet (confirmed);
            end
        end
    end}
writeln
end
end;
WriteFString (setexists, s);
writeln
end;
writeln
end;
### compress ###

Removes blanks from a 30 character string, and left justifies the result in another 30 character string. Used for validating UNIX system filenames for file access in other user's accounts.

Called by:
- GlobalDelete : when a meeting change applies to several members.
- Request, Answer : miscellaneous read and write of other user's calendar files.

```pascal
procedure compress (expanded : string30; var compressed : string30);
var
  i,j : 1..30;
begin
  compressed := ' ';
  j := 1;
  for i := 1 to 30 do
    begin
      if expanded[i] <> ' ' then
        begin
          compressed [j] := expanded [i];
          j := succ (j)
        end
    end;
  for i := j to 30 do
    compressed [i] := ' '
end;
```
{##### Check For Schedule Conflicts #####}
{
Before insertions are allowed, the beginning and ending times entered by the
user are checked against the current schedule for conflicts.
Called by
  insert       : after the times for the desired insertion are entered.
  AttemptInsert : for each date of a particular weekday present on the calendar
                  files, when inserting to the general weekday's schedule.
Input parameters
  ar           : array of time pairs - beginning and ending times for a
                 particular day's schedule.
  ArLength     : array length.
  ThisTimePair : the beginning and ending times to be inserted.
Output parameters
  ok           : true of no conflicts.
  InsertLocn   : insert location, only if ok. The location in the array
                 where ThisTimePair should be inserted to retain order.
  ConflictWith : only if not ok. The beginning and ending times of the
                 appointment for which there exists a conflict with the
                 times of ThisTimePair, the times of the insertion attempt.
}
procedure CheckForScheduleConflicts ( ar      : TimePairArray;
                                      ArLength : FileLengthRange;
                                      ThisTimePair : TimePair;
                                      var ok     : boolean;
                                      var InsertLocn : FileLengthRange;
                                      var ConflictWith : TimePair );

var
  i       : FileLengthRange;
beginn
  ok := true;
  i := 1;
  if ArLength > 0 then
  begin
    while (i<ArLength) and (ar[i].BeginTime < ThisTimePair.BeginTime) do
      i := succ (i);
    if ar[i].BeginTime = ThisTimePair.BeginTime then
      begin
        ok := false;
        ConflictWith := ar[i]
      end
    else if ar[i].BeginTime < ThisTimePair.BeginTime then
      begin
        if ar[i].EndTime > ThisTimePair.BeginTime then
          begin
            ok := false;
            ConflictWith := ar[i]
          end
        end
      else if ar[i].BeginTime < ThisTimePair.EndTime then
        begin
          ok := false;
          ConflictWith := ar[i]
        end
    else if i > 1 then
begin
  if ar[i-1].EndTime > ThisTimePair.BeginTime then
    begin
      ok := false;
      ConflictWith := ar[i-1]
    end
  end
else if i < ArLength then
begin
  if ar[i].BeginTime < ThisTimePair.EndTime then
    begin
      ok := false;
      ConflictWith := ar[i]
    end
  end;
if ok then InsertLocn := i
end;
Get FileName

General input procedure for dates, month+dates, weekdays. Following
prompts for day/date selection, this procedure reads and interprets the
user's input, and returns information that enables the calling program
to read from the corresponding files. Three input possibilities exist:
(1) date only, only allowable for short-term dates (eg. '18'),
(2) month+date, short-term or long-term dates (eg. 'Nov18'),
(3) weekday, for general weekday schedules; i.e., appointment apply to all
successive weeks on the same weekday (eg 'Wed').

Called by
  UpdateOrObserve : to know which file to read, in order to observe or edit.
  Request : to know which file on everybody's account to
            read, and then send a meeting request to.

External
  ReadString : procedure to input a string from the user's console, in
               which the information is temporarily stored.

Input parameters
  dset : set of short-term days, so the procedure knows which dates
         it can allow to be entered without a month name.
  today : pseudo-constant; today's date, the FileName of which is
          to be returned if a null line is entered.

Output parameters
  AppliesWeekly : true if a weekday is entered (eg. 'Wed').
  LongTerm : true if a month and date entered are longer than 14 days
             from the present day.
  FileName : five character string of the name of the external file
             that can be read from.

procedure GetFileName (  dset          : DaysSet;
                          today        : DaysOfMonth;
                          var AppliesWeekly : boolean;
                          var LongTerm    : boolean;
                          var DateNum     : DaysOfMonth;
                          var FileName    : string5 );

var
  ok : boolean;
  temp,i : integer;
  s : string;
  tempmonth:triple;
begin {GetFileName}
  AppliesWeekly := false;
  LongTerm := false;
  repeat
    ok := true;
    LongTerm := false;
    ReadString (s);
    readln;
    for i := 1 to 3 do FileName[i] := s[i];
    FileName[5] := ' ';
    if FileName[1] = '#' then
      begin
        DateNum := today;
        FileName[1] := chr (today div 10 + ord('0'));
      end
    else
      begin
        tempmonth := gmt.
        for i := 1 to 3 do tempmonth[i] := s[i];
      end
      begin
        tempmonth := gmt.
        for i := 1 to 3 do tempmonth[i] := s[i];
      end
      begin
        DateNum := today;
        FileName[1] := chr (today div 10 + ord('0'));
      end
    end
  until ok
end {GetFileName}
FileName[2] := chr (today mod 10 + ord('0'));
FileName[3] := ' '
end
else if FileName[1] in ['0'..'9'] then
begin
  temp := ord(FileName[1]) - ord('0');
  if FileName[2] in ['0'..'9'] then
    temp := 10*temp + ord(FileName[2]) - ord('0')
  else
    begin
      FileName[2] := FileName[1];
      FileName[1] := '0'
    end;
  FileName[3] := ' ';
  if temp <= 31 then
begin
    DateNum := temp;
    if not (DateNum in dset) then
      begin
        ok := false;
        writeln ('A file does not exist for ',DateNum:2,''.');
      end
    else
      begin
        ok := false;
        writeln ('Months have a 31 day max. ')
      end
    end
else
begin
  if (FileName = 'Sun ') or (FileName = 'sun ') or
  (FileName = 'Mon ') or (FileName = 'mon ') or
  (FileName = 'Tue ') or (FileName = 'tue ') or
  (FileName = 'Wed ') or (FileName = 'wed ') or
  (FileName = 'Thu ') or (FileName = 'thu ') or
  (FileName = 'Fri ') or (FileName = 'fri ') or
  (FileName = 'Sat ') or (FileName = 'sat ') then
begin
  AppliesWeekly := true;
  if ord(FileName[1]) > ord('Z') then
    FileName[1] := chr(ord(FileName[1]) - (ord('a') - ord('A')));
  end
else
begin
  if (FileName = 'Jan ') or (FileName = 'jan ') or
  (FileName = 'Feb ') or (FileName = 'feb ') or
  (FileName = 'Mar ') or (FileName = 'mar ') or
  (FileName = 'Apr ') or (FileName = 'apr ') or
  (FileName = 'May ') or (FileName = 'may ') or
  (FileName = 'Jun ') or (FileName = 'jun ') or
  (FileName = 'Jul ') or (FileName = 'jul ') or
  (FileName = 'Aug ') or (FileName = 'aug ') or
  (FileName = 'Sep ') or (FileName = 'sep ') or
  (FileName = 'Oct ') or (FileName = 'oct ') or
  (FileName = 'Nov ') or (FileName = 'nov ') or
  (FileName = 'Dec ') or (FileName = 'dec ') then
(FileName = 'Dec ') or (FileName = 'dec ') then
begin
  LongTerm := true;
  if ord (FileName[1]) > ord ('Z') then
    FileName[1] := chr(ord(FileName[1]) - (ord('a') - ord('A')));
    i := 4;
  while (s[i] <> '#' and (not (s[i] in ['0'..'9']))) do
    i := succ (i);
  if s[i] = '#' then
    begin
      ok := false;
      writeln ('Date must accompany month.');
    end
  else
    begin
      temp := 0;
      repeat
        temp := 10*temp + ord (s[i]) - ord ('0');
        i := succ (i);
      until not (s[i] in ['0'..'9']);
      if (temp > 31) or ((FileName = 'Apr ') or (FileName = 'Jun ') or (FileName = 'Sep ') or (FileName = 'Nov ') and (temp > 30) or (FileName = 'Feb ') and (temp > 29) then
        begin
          ok := false;
          writeln (FileName,' does not have',temp:4,' days.');
        end
      else
        begin
          DateNum := temp;
          FileName[4] := chr(ord(DateNum div 10) + ord ('0'));
          FileName[5] := chr(ord(DateNum mod 10) + ord ('0'));
          if DateNum in dset then
            begin
              for i := 1 to 3 do tempmonth[i] := FileName[i];
              if dar[DateNum].month = tempmonth then
                begin
                  LongTerm := false;
                  for i := 1 to 2 do FileName[i] := FileName[i+3];
                  for i := 3 to 5 do FileName[i] := ','
                end
            end
        end
    end
end
else
begin
  ok := false;
  writeln (FileName,' is not a valid weekday or month.');
end
end; {GetFileName}
{### Check Existence ###}  

Before a FileName representing a LongTerm date can be used to reset the corresponding file, the existence of the external file has to be insured, to prevent a run-time error. If the FileName exists as a record in the external directory file, then it is safe to read from the file. If, instead, the FileName is not found in the directory, the day-of-the-week of the LongTerm date is determined, and the corresponding weekday file is read instead, the appearance to the user is as if the LongTerm file does exist.

External

MthNum : function that returns the number of the month. Needed to deduce the year of the LongTerm date, for the DayOfWeek proc.

DayOfWeek : if a LongTerm file does not exist, then a weekday file is going to be read from instead. Which weekday file is determined by invocation of the procedure, a day-of-week algorithm.

Input parameters

FileName : The string for which directory existence is to be checked.

DateNum : the numeric value of the LongTerm date.

Output parameters

buffer : an array containing the contents of the external file 'direct' with the addition of the new LongTerm member. Only returned if the FileName is not found in the original directory. If any changes are made to the new LongTerm schedule, this array will be copied over the external file, making a confirmed new entry to the set of LongTerm dates existing on external files.

DirLength : buffer length.

found : true if the FileName is found in the directory.

ThisDay,

ThisMonth : information derived from the FileName.

}

procedure CheckExistence ( FileName : string5;
DateNum : DaysOfMonth;
var buffer : DirRecArray;
var DirLength : DirLengthRange;
var found : boolean;
var ThisDay : triple;
var ThisMonth : triple;
var ThisYear : integer );

var

index : DirLengthRange;

begin

found := false;
for index := 1 to 3 do ThisMonth [index] := FileName [index];
if (MthNum (dar[today].month) > MthNum (ThisMonth)) or
   (MthNum (dar[today].month) = MthNum (ThisMonth)) and
   (DateNum < today) then
   ThisYear := dar[today].year + 1
else
   ThisYear := dar[today].year;
DirLength := 0;
while (not eof(direct)) and (not found) do
begin
   DirLength := succ (DirLength);
   read (direct, buffer [DirLength]);
   found := buffer [DirLength].MonthDate = FileName
end;
if found then
  ThisDay := buffer [DirLength].Day
else
begin
  ThisDay := DayOfWeek (DateNum, ThisMonth, ThisYear);
  DirLength := succ (DirLength);
  buffer [DirLength].Day := ThisDay;
  buffer [DirLength].MonthDate := FileName;
end
end;
{**** Successor ****}
{
Finds the following date or day, given the current one.
Called by
  UpdateOrObserve : upon 'S' choice.
Input parameters
  AppliesWeekly : indicates the next weekday is desired.
  ThisYear : used in leap year calculation.
  LongTerm : true if long term.
  DateNum : current date (numeric).
  FileName : current filename.
Output parameters
  Longterm : possibly changed, if current date is last short-term date.
  DateNum : next date in line, after the current date.
  FileName : new Filename.
}

procedure Successor ( AppliesWeekly : boolean;
                      ThisYear : integer;
                      var LongTerm : boolean;
                      var DateNum : DaysOfMonth;
                      var FileName : string5 );

var
  i : 1..3;
  temp : triple;
  ThisMonthNum : 1..12;
  ThisMonthLength : DaysOfMonth;
begin
if AppliesWeekly then
begin
  for i := 1 to 3 do temp[i] := FileName[i];
  increment (temp);
  for i := 1 to 3 do FileName[i] := temp[i]
else
begin
  if LongTerm then
  begin
    for i := 1 to 3 do temp[i] := FileName[i]
  end
else
  begin
    temp := dar[DateNum].month;
    ThisMonthNum := MthNum(temp);
    if ThisMonthNum in [4,6,9,11] then
      ThisMonthLength := 30
    else if (ThisMonthNum = 2) and (ThisYear mod 4 = 0) then
      ThisMonthLength := 29
    else if ThisMonthNum = 2 then
      ThisMonthLength := 28
    else
      ThisMonthLength := 31;
    DateNum := succ (DateNum mod ThisMonthLength);
    if DateNum = 1 then increment (temp);
    LongTerm := LongTerm or (not (DateNum in dset));
  end
begin
  for i := 1 to 3 do FileName[i] := temp[i];
  FileName[4] := chr(DateNum div 10 + ord('0'));
    FileName[5] := chr(DateNum mod 10 + ord('0'))
    end
  else
    begin
      FileName[1] := chr(DateNum div 10 + ord('0'));
      FileName[2] := chr(DateNum mod 10 + ord('0'))
    end
  end
end;
{#### insert, Template Insert, Global Insert ####}
{
Given a particular day's schedule, these procedures perform insertions of new
appointments. Procedure 'insert' handles insertion to a single schedule, if
an insertion is to apply to several files, i.e., the schedule edited is a
general weekday schedule, then 'Template Insert' is called to apply the inser
insertion to all the required schedules.
Insert called by
UpdateOrObserve : upon an 'I' command from the user's terminal.
TemplateInsert called by
 Insert : if the schedule for which an insertion is made is a
general weekday schedule.
AttemptInsert called by
TemplateInsert : performs the actual insertion onto the schedule array.
External
  ReadFromFile,
  WriteToFile : file I/O during Template Insert.
  CheckForScheduleConflicts : before any insertion is made.
  f            : general external file of a date's schedule.
Input parameters
  ar            : the schedule (array) to which the insertion attempt is to
   be made.
  l             : array length.
  AppliesWeekly : true is the schedule is for a general weekday. Implies that
    TemplateInsert will be called to attempt the insertion to
    several date's schedules.
  dset         : set of short-term dates, so that TemplateInsert will know
    which short-term dates files to read and attempt insert.
  dar          : info about short-term dates, so that TemplateInsert will
    know which of the short-term dates are of a particular
    weekday.
  ThisDay      : The weekday for insertion, only if AppliesWeekly.
Output parameters
  ar            : the schedule if an insertion was successful.
  l             : array length, (incremented if the schedule was inserted to).
  ChangesMade  : true is an insertion was successful, implies that the new
    schedule (array) should be written over the old file.
}
procedure insert ( var ar       : RecoArray;
                    var l       : FileLengthRange;
                    var ChangesMade : boolean;
                    var AppliesWeekly : boolean;
                    var dset     : DaysSet;
                    var dar      : DaysArray;
                    var LongTerm : boolean;
                    var ThisDay  : triple );

var
  ok   : boolean;
  ThisTimePair,ConflictWith : TimePair;
  ScheduleTimes : TimePairArray;
  i,j   : FileLengthRange;

procedure TemplateInsert;
var
  ThisDate : DaysOfMonth;
ThisMonth : triple;
buffer : DirRec;
Schedule : RecArray;
ScheduleLength, InsertLocn, index : FileLengthRange;

procedure AttemptInsert (LongTerm : boolean);
begin {attempt Insert}
for index := 1 to ScheduleLength do
begin
  ScheduleTimes [index].BeginTime := Schedule [index].b;
  ScheduleTimes [index].EndTime := Schedule [index].e
end;
CheckForScheduleConflicts (ScheduleTimes, ScheduleLength, ThisTimePair,
  ok, InsertLocn, ConflictWith);
if ok then
begin
  ScheduleLength := succ (ScheduleLength);
  if ScheduleTimes [ScheduleLength-1].BeginTime >
    ThisTimePair.BeginTime then
begin
  for index := ScheduleLength downto InsertLocn+1 do
    Schedule [index] := Schedule [index-1]
end
else
  InsertLocn := ScheduleLength;
  Schedule [InsertLocn] := ar[i];
if LongTerm then
  rewrite (f,buffer.MonthDate)
else
  rewrite (f,dar[ThisDate].datechar);
WriteToFile (Schedule, ScheduleLength);
with ThisTimePair do
  writeln (tme(BeginTime),'-',tme(EndTime),' appointment ','inserted to ','ThisDay',' ','ThisMonth,ThisDate:3','.')
end
else
begin
  with ConflictWith do
    writeln ('Cannot insert to ','ThisDay',' ','ThisMonth,ThisDate:3','
      due to previous ','tme(BeginTime),'-','
      tme(EndTime),' appointment. ')
end
end; {Attempt Insert}

begin {Template Insert}
  writeln;
  for ThisDate := 1 to 31 do
  if ThisDate in dset then
    if dar[ThisDate].day = ThisDay then
      begin
        reset (f,dar[ThisDate].datechar);
        ThisMonth := dar[ThisDate].month;
        ReadFromFile (Schedule, ScheduleLength);
        AttemptInsert (false); {not LongTerm}
      end; {for}
  reset (direct);
while not eof (direct) do
begin
read (direct, buffer);
if buffer.Day = ThisDay then
begin
  for index := 1 to 3 do ThisMonth[index] := buffer.MonthDate[index];
  ThisDate := (ord (buffer.MonthDate[4]) - ord ('0')) * 10 +
  ord(buffer.MonthDate[5]) - ord ('0');
  reset (f, buffer.MonthDate);
  ReadFromFile (Schedule, ScheduleLength);
  AttemptInsert (true) {LongTerm}
end {if same weekday}
end {while not eof}
end; {Template Insert}

begin {insert}
  for i := 1 to l do
  begin
    ScheduleTimes[i].BeginTime := ar[i].b;
    ScheduleTimes[i].EndTime := ar[i].e
  end;
  if eoln then
    write ('Enter beginning and ending times. -> ');
  ok := true;
  ReadTimePair (ThisTimePair.BeginTime, ThisTimePair.EndTime);
  ThisTimePair.BeginTime := scale (ThisTimePair.BeginTime);
  ThisTimePair.EndTime := scale (ThisTimePair.EndTime);
  readln;
  CheckForScheduleConflicts (ScheduleTimes, l, ThisTimePair,
    ok, i, ConflictWith);
  if not ok then
    writeln (tme(ThisTimePair.BeginTime), ' - ',
      tme(ThisTimePair.EndTime),
      ' conflicts with previous ',
      tme(ConflictWith.BeginTime),
      ' - ',
      tme(ConflictWith.EndTime), ' appointment. ')
  else
    begin
      ChangesMade := true;
      l := succ (l);
      if ar[l-1].b > ThisTimePair.BeginTime then
        for j := 1 downto i+1 do ar[j] := ar[j-1]
      else
        i := l;
      with ar[i] do
        begin
          b := ThisTimePair.BeginTime;
          e := ThisTimePair.EndTime;
          t := other;
          setexists := false;
          writeln ('Enter description of appointment. ');
          ReadString (s);
          readln;
          if AppliesWeekly then TemplateInsert
        end;
        if AppliesWeekly then
          writeln ('Insertion made to general ',
            ThisDay, ' schedule.')
      else
writeln ('Insertion made.');
writeln
end
end; {insert}
{**** Delete, Template Delete, Attempt Delete, Global Delete ****}

Procedure for deletion of appointments / meeting on a user's personal calendar. If the deletion is to a weekday's schedule, then TemplateDelete is called to apply the deletion to all schedules of the same weekday. If the deletion affects other members, i.e., the deletion is of a meeting with several members, then GlobalDelete is called to either delete the meeting from all of the member's schedules (if the user is the owner of the meeting), or remove the user's name from the set of members on the other member's schedules (if the user is not the owner of the meeting).

Delete called by
UpdateOrObserve : upon a 'D' command entered by the user.
TemplateDelete called by
Delete : if the deletion is to apply to several weeks.
AttemptDelete called by
TemplateDelete : to perform the actual deletion and file I/O.
GlobalDelete called by
Delete : if the entry to delete is a meeting.

External
ReadStreamFile, WriteToFile : general file I/O.
compress : to format UNIX filenames for validation.
MailAr : array of local text files, to which letters are written and sent at program termination. When a deletion affects another member of a meeting, mail is sent informing the other member.

}

procedure delete ( var ar : RecArray; var l : FileLengthRange; var ChangesMade : boolean; AppliesWeekly : boolean; dset : DaysSet; dar : DaysArray; LongTerm : boolean; ThisDay, ThisMonth : triple; FileName : string5; DateNum : DaysOfMonth );

var
dtime,i,j : integer;

procedure TemplateDelete;
var
   ThisDate : DaysOfMonth;
   ThisMonth : triple;
   buffer : DirRec;
   Schedule : RecArray;
   ScheduleLength, index, n : FileLengthRange;

procedure AttemptDelete (LongTerm : Boolean);
begin {Attempt Delete}
   if ScheduleLength = 0 then
      with ar[i] do
      begin
         writeln (tme(b),' - ',tme(e),'appointment not found on ',

ThisDay,' ,',ThisMonth,ThisDate:3,' .')

end
else
begin
index := 1;
while (index < ScheduleLength) and (Schedule [index].b < dtime) do
  index := succ (index);
if (Schedule [index].b = ar[i].b) and
  (Schedule [index].e = ar[i].e) and
  (Schedule [index].s = ar[i].s) then
begin
  with Schedule [index] do
  begin
    writeln (tme(b),' - ',tme(e),', appointment deleted from ',
      ThisDay,' ,',ThisMonth,ThisDate:3,' .')
    for n := index to ScheduleLength-1 do
      Schedule [n] := Schedule [n+1];
    ScheduleLength := pred (ScheduleLength);
    if LongTerm then
      rewrite (f,buffer.MonthDate)
    else
      rewrite (f,dar[ThisDate].datechar);
    WriteToFile (Schedule, ScheduleLength)
  end
end
end
else
begin
  with ar[i] do
  begin
    writeln (tme(b),' - ',tme(e),', appointment not found on ',
      ThisDay,' ,',ThisMonth,ThisDate:3,' .')
  end
end
end;

begin {Template Delete}
  writeln;
  for ThisDate := 1 to 31 do
    if ThisDate in dset then
      if dar [ThisDate].day = ThisDay then
        begin
          reset (f,dar[ThisDate].datechar);
          ThisMonth := dar [ThisDate].month;
          ReadFromFile (Schedule, ScheduleLength);
          AttemptDelete (false)
        end; {for}
  reset (direct);
  while not eof (direct) do
  begin
    read (direct,buffer);
    if buffer.Day = ThisDay then
      begin
        for index := 1 to 3 do ThisMonth[index] := buffer.MonthDate[index];
        ThisDate := (ord (buffer.MonthDate[4]) - ord ('0')) * 10 +
                    ord (buffer.MonthDate[5]) - ord ('0');
reset (f,buffer.MonthDate);
ReadFromFlie (Schedule, ScheduleLength);
AttemptDelete (true)
end {if weekday matches}
end {while not eof}
end; [Delete Template]

procedure GlobalDelete;
var
  FName : array [users] of string30;
  ThisMember : users;
  TempAr : RecArray;
  TempArLength, TempI, TI : FileLengthRange;
  index : 1..5;

procedure init;
var
  ThisMember : users;
  i : 1..8;
begin
  for ThisMember := FirstMember to LastMember do
  begin
    FName [ThisMember] := '/usr/caldir/';
    for i := 1 to 8 do
      FName [ThisMember][i+5] := CvtName [ThisMember][i];
      compress (FName[ThisMember], FName[ThisMember])
    end
  end;

begin {Global Delete}
  init;
  for ThisMember := FirstMember to LastMember do
    if ThisMember <> ThisUser then
      if ThisMember in ar[i].uset then
        begin
          for index := 1 to 5 do
            FName[ThisMember][index+21] := FileName[index];
            compress (FName[ThisMember], FName[ThisMember]);
            reset (f, FName[ThisMember]);
            ReadFromFlie (TempAr, TempArLength);
            TempI := 1;
            while (TempI < TempArLength) and (TempAr[TempI].b < ar[i].b) do
              TempI := succ (TempI);
            if (TempAr[TempI].b = ar[i].b) and (TempAr[TempI].s = ar[i].s) then
              if TempAr[TempI].setexists then
                begin
                  if ThisUser = ar[i].owner then
                    begin
                      TempArLength := pred (TempArLength);
                      for TI := TempI to TempArLength do
                        TempAr[TI] := TempAr[TI+1];
                      writeln (MailAr[ThisMember].message,'Meeting of ',
                        ThisDay', ',ThisMonth,DateNum;3,' canceled.');
                      MailAr[ThisMember].WrittenTo := yes
                    end
                else
      end

end;
begin
  TempAr[TempI].uset := TempAr[TempI].uset - [ThisUser];
  TempAr[TempI].confirmed := TempAr[TempI].confirmed - [ThisUser];
  if ThisMember = ar[i].owner then
  begin
    writeln (MailAr[ar[i].owner].message,'Unable to attend ',
             ThisDay,', ',ThisMonth,DateNum:3,' meeting.');
    MailAr[ar[i].owner].WrittenTo := yes
  end
  end;
  rewrite (f, FName[ThisMember]);
  WriteToFile (TempAr, TempArLength)
end
end; {Global delete}

begin {delete}
  if l = 0 then
  begin
    writeln (Schedule is empty. Cannot delete.'
  end
  else
  begin
    if eoln then
    write (Enter beginning time of appointment to delete. -> ');
    ReadSingleTime (dtme);
    readln;
    dtme := scale (dtme);
    i := 1;
    while (i < l) and (ar[i].b < dtme) do i := succ(i);
    if ar[i].b = dtme then
    begin
      ChangesMade := true;
      if AppliesWeekly then
      begin
        TemplateDelete;
        writeln (tme(ar[i].b),' - ',tme(ar[i].e),', appointment deleted',
                 ' form general ',ThisDay,' schedule.'
      end
    else
      writeln ('Deletion made.');
      if ar[i].setexists then GlobalDelete;
      for j := i to l-1 do ar[j] := ar[j+1];
      l := pred(l);
      writeln
    end
  else
    writeln ('Appointment with ',tme(dtme),', beginning time ',
             'not found.')
  end
end;
{ **** Update Or Observe ****}

Procedure called by the main program to handle all observation and possibly editing of, a user's personal calendar files.

External
  Insert : procedure to insert an appointment into a schedule.
  Delete : procedure to delete an appointment/meeting from a schedule.
  GetFileName : procedure to get a date or weekday from the user's terminal.
  CheckExistenceOf : procedure to determine how to read a LongTerm date's schedule. If it doesn't already exist, the corresponding weekday's schedule is read instead.

  ReadFromFile,
  WriteToFile : general file I/O.
  WriteDirToFile : directory file output.
  f : general external file for a particular date's schedule.
  direct : external file containing a directory of LongTerm dates.

Input parameters (all used by GetFileName)
  dset : set of short term dates.
  dar : information about the 14 short-term dates.
  today : numeric pseudo-constant. Today's date.

}

procedure UpdateOrObserve (dset:DaysSet; dar:DaysArray; today:DaysOfMonth);
label
  3,4,5;
var
  FileName : string5;
  ThisDay, ThisMonth : triple;
  ThisYear : integer;
  DateNum : DaysOfMonth;
  ar : RecArray;
  l : FileLengthRange;
  AppliesWeekly, LongTerm, FileExists, ChangesMade : boolean;
  Directory : DirRecArray;
  DirLength : DirLengthRange;
  code : char;
  index : 1..3;
begin { Update Or Observe }
  3: write ('<ret> for today's schedule, enter date or day.--> ');
  4: GetFileName (dset, today, AppliesWeekly, LongTerm, DateNum, FileName);
  5: ChangesMade := false;
     FileExists := true;
     if LongTerm then
       begin (direct);
         CheckExistence (FileName, DateNum, Directory, DirLength, FileExists,
          ThisDay, ThisMonth, ThisYear);
       end
     else if AppliesWeekly then
       for index := 1 to 3 do ThisDay [index] := FileName [index]
     else
       begin
         ThisDay := dar [DateNum].day;
         ThisMonth := dar [DateNum].month;
         ThisYear := dar [DateNum].year
       end;
if FileExists then
    reset(f, FileName)
else
    reset(f, ThisDay);
ReadFromFile(ar, l);
if AppliesWeekly then
    begin
        for index := 1 to 3 do ThisDay[index] := FileName[index];
        WriteArray (ar, l, true, ThisDay, ThisDay, 0, 0)
    end
else
    WriteArray (ar, l, false, ThisDay, ThisMonth, DateNum, ThisYear);
repeat
    write ('(I)nsert, (D)elete, (S)uccessor, (L)ist, (N)ewday, or e(X)it? -> ');
    read (code);
    if code in ['I','i','D','d','S','s','L','l','N','n','X','x'] then
        case code of
        'I','i' : insert (ar, l, ChangesMade, AppliesWeekly, dset, dar, LongTerm, ThisDay);
        'D','d' : delete (ar, l, ChangesMade, AppliesWeekly, dset, dar, LongTerm, ThisDay, ThisMonth, FileName, DateNum);
        'L','l' : begin
            readln;
            if AppliesWeekly then
                WriteArray (ar, l, true, ThisDay, ThisDay, 0, 0)
            else
                WriteArray (ar, l, false, ThisDay, ThisMonth, DateNum, ThisYear)
        end;
        'N','n','S','s','X','x' :
        begin
            if ChangesMade then
                begin
                    rewrite (f, FileName);
                    WriteToFile (ar, l);
                    if LongTerm and (not FileExists) then
                        begin
                            rewrite (direct);
                            WriteDirToFile (Directory, DirLength);
                            writeln (execute, 'chmod a+rw ', FileName)
                        end
                end;
            if code in ['N','n'] then
                begin
                    if eoln then
                        begin
                            readln;
                            goto 3
                        end
                    else
                        goto 4
                end
        end
        else if code in ['S','s'] then
        begin
            readln;
            Successor (AppliesWeekly, ThisYear, LongTerm,
DateNum, FileName);
goto 5
end
end
end
else begin writeln ('Invalid code'); readln end
until code in ['X','x'];
readln;
end; [Update Or Observe]
**** Make Or Answer Requests ****
{
A procedure logically divided into two nested procedures: 'Request' & 'Answer'.
Other procedures included are 'Init', 'BlackOutConflicts', 'DisplayPossibilities'
and function 'Conflict'.
Input parameters
    dset    : set of short-term dates, used by 'GetFileName'.
    dar     : information about the short-term dates.
    today   : today's date.
}
procedure MakeOrAnswerRequest (dset:DaysSet; dar:DaysArray; today:DaysOfMonth);
type
    InfoRec = record
        FileExistsT : FakeBoolean;  {information about each user's }
        FName : string30;         {--external files -- in terms of}
        Weekday : string30;       {this user. All FileNames are }
        DirName : string30;       {--of length 30 to accommodate }
        Directory : DirRecArray;  {--Unix directory and file naming}
        DirLength : DirLengthRange
    end;
    InfoArray = array [users] of InfoRec;
    BArray = array [0..287] of boolean;  {true if the time is a possibility}
                                   {for a meeting, false otherwise. }
var
    members : UserSet;
    ThisMember : users;
    TimeSpan : integer;
    Available : BArray;
    info : InfoArray;
    Ar : RecArray;
    ArLength : FileLengthRange;
    AppliesWeekly, LongTerm, Standard, FileExists : boolean;
    FileName : string5;
    ThisDay, ThisMonth : triple;
    ThisYear : integer;
    c, code : char;
    ok : boolean;
    DateNum : DaysOfMonth;
    index : integer;
    Times : TimePairArray;
    ThisTimePair : TimePair;
    st : string;
{** init **}
{
Re-assigns the values of the information array to external filename stubs.
The string values are left justified, the right 'halves' are assigned in
the calling programs, and the strings are compressed. The resulting strings
are UNIX system filenames, on other user's accounts.
External
    info : information array (of records).
}
procedure init;
var
    ThisMember : users;
    i : 1..8;
begin
    for ThisMember := FirstMember to LastMember do
        with info [ThisMember] do
            begin
                FName := '/usr/   /caldir/';
                DirName := '/usr/   /caldir/direct';
                for i := 1 to 8 do
                    begin
                        FName [i+5] := CvtName [ThisMember][i];
                        DirName[i+5] := CvtName[ThisMember][i]
                    end;
                compress (DirName, DirName)
            end;
end;

{** Black Out Conflicts **}
{
The boolean array 'Available', initially set to true, is sent to this
procedure with an array of time pairs. The time pairs represent unavailable
times, thus the elements of 'Available' which correspond to those times are
set to false. After this procedure has been called with every member's
schedule times, the true valued elements of 'Available' represent free times.
Called by Request.
Input parameters
    Times : array of time pairs representing the unavailable times for
            a particular member.
    length : array length.
    Available : the boolean array before 'blacking out' from the member.
Output parameter
    Available : the boolean array after 'blacking out' from the member.
}
procedure BlackOutConflicts (Times:TimePairArray; Length:FileSizeRange;
                              var Available:BArray);
var
    index : FileSizeRange;
    NotHere : 0..287;
begin
    for index := 1 to Length do
        for NotHere := Times [index].BeginTime to Times [index].EndTime-1 do
            Available [NotHere] := false
end;
[* Display Possibilities *]
{
When the boolean array 'Available' has been 'blackened out' with all the
member's schedules, the times-slots available that are equal to or greater
than the required time-span (entered by the user) are displayed and stored
in an array of time pairs.
Called by
Request
Input parameters
    Available    : the boolean array.
    TimeSpan     : minimum time span required, in minutes.
    DateNum,
    ThisDay,
    ThisMonth   : used for a display heading.
Output parameters
    Times    : array of time pairs, representing meeting possibilities.
    length   : array length.
}
procedure DisplayPossibilities ( Available    : BArray;
    TimeSpan     : integer;
    DateNum      : DaysOfMonth;
    ThisDay,
    ThisMonth   : triple;
    var Times    : TimePairArray;
    var Length   : FileLengthRange);

var
    index : 0..287;
begin
    Length := 0;
    index := 0;
    while index < 287 do
        begin
            while (not Available[index]) and (index < 287) do index := succ(index);
            Times [Length+1].BeginTime := index;
            while (Available[index]) and (index < 287) do index := succ (index);
            Times [Length+1].EndTime := index;
            with Times [Length+1] do
                if (EndTime - BeginTime) >= TimeSpan then Length := succ (Length)
        end;
    writeln ('Here are the possible free times for ',ThisDay,'
        ThisMonth,DateNum:3,':
        writeln;
        for index := 1 to Length do
            with Times [index] do
                begin
                    write (' ':8,tme(BeginTime),'- ',tme(EndTime),' ':8);
                    if (index mod 2) = 0 then writeln
                end;
                if (index mod 2) <> 0 then writeln
end;
{** Conflict **}

Function 'Conflict' checks an array of time pairs and a single time pair and returns true if an insertion of the single time pair can be made, false otherwise.
Called by
Request.
Input parameters
  - Times : array of time pairs.
  - length : array length.
  - ThisTimepair : the time pair for which an insertion is desired.
}

function Conflict ( Times : TimePairArray;
  Length : FileLengthRange;
  ThisTimePair : TimePair ) : boolean;

var
  index : FileLengthRange;
  temp : boolean;

begin
  temp := true;
  for index := 1 to Length do
    begin
      if (Times [index].BeginTime <= ThisTimePair.BEGIN) and
         (Times [index].EndTime   >= ThisTimePair.ENDING) then
        temp := false;
    end;
  if temp then
    begin
      with ThisTimePair do
        writeln (tme(BEGIN),' - ',tme(ENDING),', not available.','
               ', 
               ' Here are the available times again:');
      writeln;
      for index := 1 to Length do
        with Times [index] do begin
          write (' ':8, tme(BEGIN),' - ',tme(ENDING),' ':8);
          if (index mod 2) = 0 then writeln
          end;
        writeln;
        if (index mod 2) <> 0 then writeln
      end;
      Conflict := temp
    end;
end;
{**** Request ****}
[
With the 'request' choice taken by the user, this procedure performs the
processes necessary to make a meeting request to any number of other users.
The user is prompted for a list of members, a date, a time-span, if the
meeting is to take place during standard (8:00-5:00) hours, and then finally,
the meeting times. At successful completion, the request is made, and mail
sent to each member informing him/her of the request.
External
Init, BlackOutConflicts, DisplayPossibilities, Conflict : described above.
ReadSet : To input the set enumerated set of members from the user's
terminal.
GetFileName : To input the date or month+date from the user's terminal.
CheckExistence : if the date is longterm, to determine where to read the
file from; the date's file or a weekday file.
ReadFromFile : file input of a date's schedule to an array.
ReadTimePair : for meeting beginning and ending times.
ReadString : for meeting description.
WriteDirToFile : if a new LongTerm date is created, the newly augmented
directory must be written over the previous one.
compress : to format UNIX system filenames.
WriteToFile : file output from an updated array.
MailArr : array of local text files - letters to be mailed.
]
procedure Request;
begin {Make Request}
  init;
  repeat
    writeln ('With whom would you like an appointment?');
    write ('Enter names or list. -> ');
    ReadSet (members);
    members := members + [ThisUser];
    readln;
    if list in members then
      begin
        for ThisMember := FirstMember to LastMember do
          write (CvtName [ThisMember]);
        writeln
      end
    until not (list in members);
  repeat
    write ('For how long? (enter minutes) -> ');
    ReadTimeSpan (TimeSpan);
    readln;
    if TimeSpan > 720 then
      writeln ('720 minute limit for meetings.')
  until TimeSpan <= 720;
  repeat
    init;
    repeat
      repeat
        write ('Enter preferred date, or month and date. -> ');
        GetFileName (dset, today, AppliesWeekly, LongTerm, DateNum, FileName);
        until not AppliesWeekly;
      write ('Standard 8:00 - 5:00? (Y/N) -> ');
      until not (c in ['Y','N'])
    end
  until TimeSpan <= 720;
end.
readln (c);
Standard := (c<>'N') and (c<>'n');
if Standard then
begin
  for index := 0 to 287 do Available[index] := false;
  for index := 84 to 191 do Available[index] := true
end
else
  for index := 0 to 287 do Available[index] := true;
for ThisMember := FirstMember to LastMember do
if ThisMember in members then
with info [ThisMember] do
begin
  FileExists := true;
  if LongTerm then
  begin
    reset (direct, DirName);
    CheckExistence (FileName, DateNum, Directory, DirLength,
                    FileExists, ThisDay, ThisMonth, ThisYear)
  end
else
  begin
    ThisDay := dar [DateNum].day;
    ThisMonth := dar [DateNum].month
  end;
if FileExists then FileExistsT := yes else FileExistsT := no;
if not FileExists then
begin
  for index := 1 to 3 do FName [index+21] := ThisDay[index];
  compress (FName, Weekday);
  reset (f, Weekday);
  ReadFromFile (Ar, ArLength);
end;
for index := 1 to 5 do FName [index+21] := FileName [index];
compress (FName, FName);
if FileExists then
begin
  reset (f, FName);
  ReadFromFile (Ar, ArLength)
end;
for index := 1 to ArLength do
begin
  Times [index].BeginTime := Ar [index].b;
  Times [index].EndTime := Ar [index].e
end;
BlackOutConflicts (Times, ArLength, Available)
end;
DisplayPossibilities (Available, TimeSpan, DateNum, ThisDay,
                      ThisMonth, Times, ArLength);
if ArLength = 0 then
begin
  write ('No free times exist. Do you want to try another day? ->');
  readln (c);
  if (c <> 'Y') and (c <> 'y') then c := 'x'
else c := 'n'
end
else
begin
    writeln;
    write ('Do you want one of these? \text{Y/N}\text{e(X)it} \rightarrow ');
    readln (c)
end;
ok := (c>'N') and (c>'n')
until ok;
if (c>'X') and (c>'x') then
begin
  repeat
    write ('Enter beginning and ending times. \rightarrow ');
    with ThisTimePair do
    begin
      ReadTimePair (BeginTime, EndTime);
      BeginTime := scale (BeginTime);
      EndTime := scale (EndTime)
    end;
    readln;
  until not Conflict (Times, ArLength, ThisTimePair);
  writeln ('Enter description of appointment. ');
  ReadString (st);
  readln;
  for ThisMember := FirstMember to LastMember do
    if ThisMember in members then
      with info [ThisMember] do
      begin
        if FileExistsT = yes then
          reset (f,FName)
        else
          reset (f,Weekday);
        ReadFromFile (Ar, ArLength);
        index := succ (ArLength);
        if index > 1 then
          while (Ar [pred(index)].b >= ThisTimePair.EndTime) and (index > 1) do
          begin
            Ar [index] := Ar [pred(index)];
            index := pred(index)
          end;
        ArLength := succ (ArLength);
        with Ar [index], ThisTimePair do
        begin
          b := BeginTime;
          e := EndTime;
          t := request;
          setexists := true;
          owner := ThisUser;
          confirmed := [ThisUser];
          uset := members;
          s := at
        end;
        rewrite (f, FName);
        WriteToFile (Ar, ArLength);
        if FileExistsT = no then
        begin
          writeln (execute, '\text{chmod a+rw }', FName);
        end;
      end;
    end;
end;
rewrite (direct, DirName);
    WriteDirToFile (Directory, DirLength)
end;
if ThisMember <> ThisUser then
begin
    MailAr[ThisMember].WrittenTo := yes;
    writeln (MailAr[ThisMember].message,'Requesting an appointment ',
        'for ',ThisDay,' ',ThisMonth,DateNum:3,'.');
end
end {for} 
end {if not exit}
end; {Make Request}
**** Answer ****

When the 'answer' choice is taken, this procedure prompts for and then
displays a day containing meeting requests. One-by-one the requests are
given in a prompt with an accept, reject, or move-on choice given to the
user. If a request is accepted, the user is added to the confirmed field
of the other members. If, instead, the user rejects the request, then his/her
name is removed from the other member's 'members' field in that particular
meeting. Either way, mail is sent to the owner of the meeting informing
him/her of the decision.

External

    GetFileNane : for the date to look at requests.
    CheckExistence : for status of a LongTerm date.
    ReadFromFile,
    WriteToFile : file I/O of particular date's schedules.
    WriteArray : terminal display of a date's schedule containing requests.
    Compress : formats filenames for UNIX.
    MailArray : array of local text files - letters to be mailed.

procedure Answer;

var

    i, index, TempArrLength : FileLengthRange;
    TempArr : RecArray;
    dummy1 : DirRecArray;
    dummy2 : DirLengthRange;
    RNum : FileLengthRange;
    RequestExists : boolean;

begin

    repeat

        write ('Enter date for which requests are to be answered. -> ');

        GetFileNane (dset, today, AppliesWeekly, LongTerm, DateNum, FileName)

        until not AppliesWeekly;

        FileExists := true;

        reset (direct);

        if LongTerm then

            CheckExistence (FileName, DateNum, dummy1, dummy2,
                            FileExists, ThisDay, ThisMonth, ThisYear)

        else

            ThisDay := dar[DateNum].day;
            ThisMonth := dar[DateNum].month;
            ThisYear := dar[DateNum].year

        end;

        if FileExists then

            begin

                reset (f,FileName);

                ReadFromFile (Ar, ArLength);

                RequestExists := false;

                for index := 1 to ArLength do
                    if (Ar[index].t = request) and (ThisUser <> Ar[index].owner) then
                        RequestExists := true

            end;

            if (not FileExists) or (not RequestExists) then
                writeln ('No Requests exist on ', FileName)
            else


begin
  WriteArray (Ar, ArLength, false, ThisDay, ThisMonth, DateNum, ThisYear);
  index := 1;
  RNum := 1;
  while index <= ArLength do
    begin
      while ((Ar[index].t <> request) or (Ar[index].owner = ThisUser))
        and (index < ArLength) do index := succ (index);
      if (Ar[index].t = request) and (Ar[index].owner <> ThisUser) then
        begin
          writeln ('R',RNum:1,'?',Acept, (R)eject, or (M)ove-on -> ');
          RNum := succ (RNum);
          readln (o);
          if c in ['A','a','R','r'] then
            begin
              write (MailAr[Ar[index].owner].message,'Appointment for ',',ThisDay, ','ThisMonth,DateNum:3);
              MailAr[Ar[index].owner].WrittenTo := yes;
              if c in ['A','a'] then
                writeln (MailAr[Ar[index].owner].message,' accepted. ')
              else
                writeln (MailAr[Ar[index].owner].message,' rejected. ');
              for ThisMember := FirstMember to LastMember do
                if ThisMember in Ar[index].uset then
                  with info[ThisMember] do
                    begin
                      init;
                      for i := 1 to 5 do FName[i+21] := FileName[i];
                      compress (FName, FName);
                      reset (f,FName);
                      ReadFromFile (TempAr, TempArLength);
                      i := 1;
                      while (TempAr[i].b <> Ar[index].b) and (i < TempArLength) do i := succ(i);
                      if TempAr[i].s = Ar[index].s then
                        with TempAr[i] do
                          begin
                            if c in ['A','a'] then
                              begin
                                confirmed := confirmed + [ThisUser];
                                rewrite (f, FName);
                                WriteToFile (TempAr, TempArLength)
                              end
                            else if c in ['R','r'] then
                              begin
                                uset := uset - [ThisUser];
                                rewrite (f, FName);
                                WriteToFile (TempAr, TempArLength)
                              end
                          end
                        end;
                      if c in ['A','a'] then
                        Ar[index].t := other
                      else if c in ['R','r'] then
                        begin
                          ArLength := pred (ArLength);
                        end
                    end
                end;
            end
        end
    end
end
for i := index to ArLength do
    Ar[i] := Ar[i+1]
end;
rewrite (f, FileName);
WriteToFile (Ar, ArLength)
end
end;

index := succ (index)
end
end;

begin {Make or Answer Requests}
    write ('(R)equest a meeting, or (A)nswer a request? -> ');
    readln (code);
    if code in ['R','r','A','a'] then
        case code of
            'R','r' : Request;
            'A','a' : Answer
        end
    else writeln ('Invalid code. Returning to main menu. ')
    end;
{******************************** ELECTRONIC CALENDAR ********************************}
{******************************** MAIN PROGRAM ********************************}
{
External
  InitGlobalVars : initialize global variables.
  DatesInfo : get information about today's date, and the 14
              short-term dates.
  UpdateOrObserve : Update Or Observe personal calendar.
  MakeOrAnswerRequests : Make or Answer Meeting Requests.
  DumpLocalMailFiles : to external files, so the mail can be sent.
}
begin
  rewrite (execute);
  InitGlobalVars;
  writeln; writeln ('Welcome to Electronic Calendar.');
  DatesInfo (dset,dar,today);
  writeln;
  with dar[today] do
    writeln ('Today is ',day,' ',month,today:3,','19',year:2,'.');
repeat
  writeln; writeln ('Main Menu:');
  write ('1. Update or Observe Personal Calendar.');
  writeln ('  2. Send or Answer Meeting Requests.');
  write ('X. Exit.');
  writeln; write ('Please Enter Command Number or X. -> ');
  readln (code);
  if code in ['1','2','X','x'] then
    case code of
      '1' : UpdateOrObserve (dset,dar,today);
      '2' : MakeOrAnswerRequest (dset,dar,today);
      'x','X' : begin
        writeln;
        writeln ('Electronic Calendar Over.');
        writeln
      end
    end
  else writeln ('Invalid code')
  until code in ['X','x'];
  DumpLocalMailFiles
end.
program NewDay (output, direct);
const
  MaxDirLength = 50;
type
  pair = packed array [1..2] of char;
  triple = packed array [1..3] of char;
  string5 = packed array [1..5] of char;
  DaysOfMonth = 1..31;
  DirRec = record
    Day : triple;
    MonthDate : string5
  end;
  DirLengthRange = 0..MaxDirLength;
  DirFileType = file of DirRec;
var
  NewDate : pair;
  NewDay, NewMonth : triple;
  FileName : string5;
  direct : DirFileType;
  index : 1..5;
  Exists : boolean;
[
**** Find New ****]

Procedure to calculate and return information about the new day, i.e., the one which will be considered short-term tomorrow. The information returned describes the date exactly two weeks from the current date, which, tomorrow will be the last date considered short-term.

Called by

NewDay's main program.

External
date : Berkeley Pascal built-in that returns current date info.

Output parameters

NewDay,
NewMonth,
NewDate : information about new date.

}

procedure FindNew (var NewDay, NewMonth : triple; var NewDate: pair);
var
  a : alfa;
  month : triple;
  i,year,datenum: integer;
  ThisMonthLength : DaysOfMonth;

function DayOfWeek (date: DaysOfMonth; month: triple; year: integer): triple;
var
  DayNum: 0..6;
  MonthNum : 1..12;
  funct : array [1..12] of integer;
begin (* function *)
  if month = 'Jan' then MonthNum := 1
  else if month = 'Feb' then MonthNum := 2
  else if month = 'Mar' then MonthNum := 3
  else if month = 'Apr' then MonthNum := 4
  else if month = 'May' then MonthNum := 5
  else if month = 'Jun' then MonthNum := 6
  else if month = 'Jul' then MonthNum := 7
  else if month = 'Aug' then MonthNum := 8
  else if month = 'Sep' then MonthNum := 9
  else if month = 'Oct' then MonthNum := 10
  else if month = 'Nov' then MonthNum := 11
  else if month = 'Dec' then MonthNum := 12
  else halt;
  if ((month = 'Jan') or (month = 'Feb')) and (year mod 4 = 0) then
    DayNum := (funct[MonthNum] + date + year + year div 4 - 1) mod 7
  else
    DayNum := (funct[MonthNum] + date + year + year div 4) mod 7;
  case DayNum of
    0 : DayOfWeek := 'Sat';
    1 : DayOfWeek := 'Sun';
    2 : DayOfWeek := 'Mon';
    3 : DayOfWeek := 'Tue';
    4 : DayOfWeek := 'Wed';
    5 : DayOfWeek := 'Thu';
procedure increment(var this: triple);
begin
  if this = 'Jan' then this := 'Feb'
  else if this = 'Feb' then this := 'Mar'
  else if this = 'Mar' then this := 'Apr'
  else if this = 'Apr' then this := 'May'
  else if this = 'May' then this := 'Jun'
  else if this = 'Jun' then this := 'Jul'
  else if this = 'Jul' then this := 'Aug'
  else if this = 'Sep' then this := 'Oct'
  else if this = 'Oct' then this := 'Nov'
  else if this = 'Nov' then this := 'Dec'
  else if this = 'Dec' then this := 'Jan'
  else halt
end;

begin (* procedure FindNew *)
  date (n);
  if n[1] = ' ' then n[1] := '0';
  datenum := 0;
  for i := 1 to 2 do
    datenum := datenum * 10 + ord(n[i]) - ord('0');
  for i := 4 to 6 do
    month[i-3] := n[i];
  year := 0;
  for i := 8 to 9 do
    year := year * 10 + (ord(n[i]) - ord('0'));
  NewDay := DayOfWeek (datenum, month, year);
  if (month = 'Apr') or (month = 'Jun') or
    (month = 'Sep') or (month = 'Nov') then
    ThisMonthLength := 30
  else if (month = 'Feb') and (year mod 4 = 0) then
    ThisMonthLength := 29
  else if (month = 'Feb') then
    ThisMonthLength := 28
  else
    ThisMonthLength := 31;
  for i := 1 to 14 do
    begin
      datenum := succ(datenum);
      if datenum > ThisMonthLength then
        begin
          increment (month);
          datenum := 1
        end;
    end;
  NewMonth := month;
  NewDate[1] := chr (datenum div 10 + ord ('0'));
  NewDate[2] := chr (datenum mod 10 + ord ('0'))
end;
{**** Check Existence ****}
{
The directory file is searched for the new month and date to see if a long-term
file exists. If it does, then (1) the external long-term file will be copies
into the short-term file, (2) the corresponding record in the directory will
be removed, and (3) the long-term file removed. If, instead, the matching
record is not found in the directory, then the corresponding weekday is copied
into the new short-term file.

Called by
   NewDay main program.

External
direct     : external file name of directory.

Output parameters
   FileName   : the filename from which to read. Might be a long-term date
                (eg. 'Nov18') or a weekday ('Thu').
   Exists     : true if the long-term date is found in the directory.
}

procedure CheckExistence (FileName: string5; var Exists: boolean);
var
   DirLength, FoundLocn, index : DirLengthRange;
   buffer : array [DirLengthRange] of DirRec;
begin
   DirLength := 0;
   reset (direct);
   Exists := false;
   while not eof (direct) do
   begin
      DirLength := succ (DirLength);
      read (direct, buffer [DirLength]);
      if buffer [DirLength].MonthDate = FileName then
      begin
         Exists := true;
         FoundLocn := DirLength
      end
   end; (* while not eof *)
   if Exists then
   begin
      DirLength := pred (DirLength);
      for index := FoundLocn to DirLength do
         buffer [index] := buffer [index+1];
      rewrite (direct);
      for index := 1 to DirLength do
         write (direct, buffer [index])
   end
end;
{**************************** NEW DAY *******************************}
{**************************** MAIN PROGRAM ***************************}
{
Called by
   UNIX system via 'at' command.
External
   FindNew : procedure to get information about new short-term date.
   CheckExistence : procedure to see if the new short-term date already
                  exists as a long-term date.
}
begin {main}
   FindNew (NewDay,NewMonth,NewDate);
   for index := 1 to 3 do FileName [index] := NewMonth [index];
   for index := 4 to 5 do FileName [index] := NewDate [index-3];
   CheckExistence (FileName, Exists);
   if not Exists then
     begin
       for index := 1 to 3 do FileName [index] := NewDay [index];
       for index := 4 to 5 do FileName [index] := ' ';
     end;
   writeln ('cp ',FileName,' ',NewDate);
   writeln ('chmod a+w-r ',NewDate);
   if Exists then
     writeln ('rm ',FileName)
end.
A "UNIX" BASED
ELECTRONIC CALENDAR SYSTEM

by

DAVID OWEN JAMES

B. A., Bethany College, Lindsborg, Kansas, 1981

AN ABSTRACT FOR A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982
ABSTRACT

Appointment calendars are time management tools that allocate specific time segments for activities on a given day. Electronic calendars are computerized versions of the more traditional paper counterparts, but with several advantages. The major advantage can be seen when scheduling meetings between several principals or users. Typically the required process involves checking each principal's schedule against the proposed meeting time until a conflict-free meeting time for all the principals involved is found. An automated version of this process can schedule such a meeting time almost immediately, for any number of principals. An implementation of such an electronic calendar system currently running on a 32-bit Perkin-Elmer 3220 computer at the KSU Department of Computer Science is described.