PREFER:

Small Group Decision Support System Tool

by

NANCY JANE CALHOUN

A.A., Graceland College, 1962
B.S., Colorado State University, 1964
B.S., Kansas State University, 1982

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

Major Professor

Table of Contents

Table of Illustrations

ACKNOWLEDGEMENTS

I am grateful to Dr. Elizabeth Unger, my major professor, who quided me through the many requirements necessary to reach my goal. Dr. David Gustafson and Dr. Maarten van Swaay assisted by being on my committee. I also appreciate the support given to me by other lecturers in the department, the staff personnel, and other graduate students.

Constanza Hagmann was very gracious to let me read her work and implement a small portion of it. After she designed the model, I was able to follow and give it more substance.

Special thanks must go to my husband who was very supportive, encouraging me, and willing to adjust to my schedule and routine. He frequently had to pull more than his fair share to make everything work while I studied.

PREFER:

A Small Group Decision Support System Tool

Introduction:

The potential of computers to he a heneficial tool is heing recognized in more and more areas of human endeavor. Our societies are hecoming more centered on information exchange, and the ahility of computers to manage information is seen as both attractive and even necessary for the future. Currently, computers are widely used as tools hy individuals participating in decision processes, hut groups making decisions are more likely to meet in conference rooms with computer printouts and without computers. The hardware features of networked computers allow groups of individuals to communicate with each other, hut the software support for effective interaction within a group has not yet appeared.

This report describes an implementation of a Small Group Decision Support System hased on a model developed hy Hagmann [HAGM88a]. The report hegins with a hrief discussion of group dynamics and of group decision theories. A short discussion of the Small Group Decision Support System (SGDSS) and its background follows. Then the specifics of this implementation (referred to hereafter as the "PREFER tool" or as "PREFER") are discussed, with details included in Appendices. This report concludes with a discussion of the anticipated usage of PREFER and further ways in which it could he developed.

Ways to benefit humanity through expanded use of the computer as a tool must be developed hecause the future of computers is now.

CHAPTER I

DECISIONS IN GROUPS

Each human must face one decision after another. If one is old enough and smart enough to read this report, one is probably already well experienced in decision making as an individual and as part of a group. In infancy, a baby may have to choose only between sleeping and eating, but later, as a member of a board of directors, that same individual may have to choose between expanding a business, staying with the status quo, moving the business to a new area, or going out of business. Some decisions have trivial consequences, while others may have profound impact.

John Dewey [DEWE10] suggests 5 phases of logical thinking:

1. awareness of the existence of a problem
2. definition of the problem
3. the suggestions of possible solutions for the problem
4. the elaboration of the suggested solutions
5. testing and evaluation of selected solution

An individual may frequently make personal decisions alone, but as the problem grows, the number of people included in the decision process usually grows also. The phases of logical thinking remain the same, but a group introduces additional complexity. The long-standing cliches about committees, such as "a camel is a horse designed by a committee", have more truth than most committees would care to admit!

Ewbank and Auer [EWBA51a] identify certain thinking patterns in individuals and describe areas where being a member of a group intensifies these individual behavioral tendencies. According to them, in a group one responds more to the leading of others and tends to conform to group standards in belief and action. One may support a decision because "everyone else is

for it". One is more likely to follow illogical arguments or succumh to personal appeal and the charisma of someone else in the group. If one has prejudices, one may have those intensified by expressions of similar prejudices in others. One may let logical-thinking take a hack seat to seeking group approval or one may more easily confuse personal desires with convictions.

The composition of a group colors one's actions and opinions. One is not likely to oppose a superior or a valued co-worker in a face-to-face meeting even if one's convictions would dictate otherwise. On the other hand, one may find oneself opposing another whose personality contrasts with one's own even when hoth actually agree on an issue. Anyone who has sat through a long and less-than-productive meeting desires ways to make group decisions easier. Businesses especially would like to get the hest decisions from management groups.

Groups reach agreement in several ways:

1. Authority - "hecause I said so"
2. Enumeration - majority rules
3. Compromise - I'll give in here if you'll give in there
4. Integration or consensus - you have convinced me that this is the hest way

Ewhank and Auer conclude that

> "...[consensus] is the ideal one; it can result most easily when group members begin the analysis of a prohlem before their opinions are firmly set.... Integration is not a likely result, however, where external pressures operate on a group, where tensions are high within a group, or where a deadline for decision must he met.... Before intelligent and effective problem-solving can result, hoth leaders and participants must understand the psychology of group hehavior and develop the sensitivities and skills appropriate to democratic group action." [EWBA51h].

Because of modern-day interest in group decision patterns, researchers are proposing many theories regarding how groups reach decisions when confronted with multiple criteria. Hagmann [HAGM88a] lists several major types of theories: social choice/group decision theory, expert judgement, game theory, and team theory.

Hagmann chose the group decision theory for her model because of its goal of consensus in the group through the use of voting, social choice function, and social welfare function. A decision in Hagmann's model is a consensus of participants with different levels of expertise so it did not match the expert judgement techniques. Game theory was rejected because of its assumption that participants are pursuing strict personal gain rather than organizational gain. Team theory was rejected because there was no effective way to generalize the gross pay-off function based on quality of information available to members of the group.

> Since the present SGDSS design is aimed at small groups operating in a situation of a medium level of conflict, the method used to achieve group consensus should provide both a means for careful analysis of individual reasoning and input and an impartial way for the group members to reduce individual preferences to a group choice. [HAGM88b]

Group Decision Theory attempts to enable all individuals of a group to be active participants in the decision process. Group Decision Support Systems (GDSS) are models utilizing computers to aid the group decision process. GDSS's can operate on three different levels: communication, group decision modeling, and machine-induced meeting patterning.

Examples of these levels are contained in Colab, a computer-supported meeting setting being developed at Xerox PARC [STEF87]. In this setting the networked computers are used to imitate the functions of a blackboard in a meeting, but here each participant can figuratively stand at the blackboard and input ideas that all can see on individual terminals (GDSS communication level). Another software process in Colab, Cognoter, imitates a meeting style for collaborative writing by organizing a meeting into three distinct phases--"brainstorming", "organizing", and "evaluation"--each of which gives emphasis to a different set of activities (GDSS meeting patterning). The group decision modeling level is part of a Small Group Deci-

sion Support System (SGDSS) developed by Hagmann. The PREFER tool is based on the SGDSS model, and is a distributed voting server.

One scenario of PREFER usage is to follow an idea-presentation meeting with time for individual research on tbe alternatives selected at that meeting. A local area network can be used by members of tbe group to enter individual research wbicb can be reviewed by everyone in the group before voting. Tbus the group ordering of tbe preferences can be done outside of a face-to-face meeting resulting in better utilization of the time of management personnel.

Another style of usage is to add PREFER to Colab. Here PREFER is a tool of the meeting environment itself. This decision tool is used to narrow brainstormed ideas into alternatives for final consideration, and then to form tbe final ordering of the alternatives selected. PREFER might be particularly beneficial witbin meetings to modify some of the group dynamic problems such as peer pressure, group think, and disruptive interpersonal relationships. Each individual may gain some freedom of expression which results in more accurate evaluation of personal preferences.

Still another PREFER usage is in polling. For example, an individual may bave several ideas tbat seem worthy of a group discussion, but does not want to spend a great deal of time in getting background material until there is some indication of interest. The individual can prepare a sketcby outline of alternatives and ask tbe group to indicated interest (take a "straw vote"). Witb indication of interest the individual can pursue gathering information about the most popular ideas before the next face-to-face meeting.

Tbe SGDSS model is described in more detail in the next chapter.

CHAPTER II

SMALL GROUP DECISION SUPPORT SYSTEM (SGDSS)

In this chapter, the aspects of the SGDSS model which are of importance to this report are discussed. SGDSS is designed as part of a Local Area Decision Network which would he utilized in a small management group setting of five to nine individuals. An integral part of this support system is an algorithm, called the fuzzy hinary relation algorithm, which is used to ohtain a group decision from individual opinions. Fuzzy set theory was chosen hy Hagmann hecause the modeling of a preference relation over a set of alternatives can hest he achieved in terms of fuzzy set theory [HAGM88a]. Hagmann's algorithm is a refinement of the fuzzy hinary relations proposed hy Blin and Whinston [BLIN74]. It fits into the decision making process at the point where several alternatives have heen defined and the group wants to order them from most-preferred to least-preferred. The overall group ordering will he hased on the individual personal preference orderings of the group.

The following excerpts from Hagmann's discussion of the algorithm introduce and define some of the terminology used in explaining the model:

> Using the group decision theory definition (see section 4.2.2), in a decision making situation we have a finite group of n individuals confronted with a set A of m alternatives over which they are to choose jointly in accordance with their individual opinions. These individual opinions are simply assumed to he n linear orderings $\{O_h\}_{1<h\leq n}$ over A. Thus the total set of all possihle linear orderings forms the symmetric group of order m! or a suhset thereof [Blin,1974].
>
> . . . .
>
> Definition 4.3: Social Preference relation
> A social preference relation R is a fuzzy suhset of A x A characterized hy a memhership function μ_R : A X A $\rightarrow$ [0,1] which associates with each pair (a_i,a_j) its grade of memhership $\mu_R(a_i,a_j)$ in R. [Blin,1974].

. . . .

Since the SGDSS has as one of its goals to maximize participation of group members, not only by counting their votes impartially to reach a group consensus but by letting them see exactly how the group vote has been computed, μ_{R_1} has been selected as the social preference relation to be implemented.

. . . .

Definition 4.5: α-level set
An α-level set (R_α) of a fuzzy relation R is a nonfuzzy set in A x A defined by [Zadeh,1971]:

$$R_\alpha = \{ (a_i,a_j) \mid \mu_{R(a_i,a_j)} \geq \alpha \}$$

These R_α sets form a nested sequence of nonfuzzy relations with $\alpha_1 \geq \alpha_2 \rightarrow R_{\alpha_1} \subset R_{\alpha_2}$.

. . . .

The α-level sets (Def.4.5) can be thought of as "agreement levels" for a group of individuals. They represent a set of thresholds for the acceptance or rejection of a certain binary preference at the societal level. As we process the binary preferences through this multi-level filter, some binary preferences become no longer socially acceptable. [HAGM88c]

The algorithm can be illustrated in this example where seven individuals order four alternatives (a,b,c,d) in the following way:

individual	ordering
#1	O_1 - {a, b, c, d}
#2	O_2 - {b, c, d, a}
#3	O_3 - {d, b, c, a}
#4	O_4 - {c, a, d, b}
#5	O_5 - {d, a, c, b}
#6	O_6 - {c, d, a, b}
#7	O_7 - {b, d, a, c}

Hagmann's algorithm tallys each vote into a matrix showing the number of times the row alternative is preferred over the column alternative. If this tally in each cell is divided by the number of authorized voters, then an "average" is obtained which is the membership function $\mu_R : A \times A \rightarrow [0,1]$. The average tallies are inserted into another matrix. The total and average matrices for the example orderings are shown in Figure 2.1.

Final total vote
(MATRIX 1 in algorithm)

Alt\Alt:	A	B	C	D
A:	0	4	3	2
B:	3	0	4	3
C:	4	3	0	4
D:	5	4	3	0

average vote:
(MATRIX 2 in algorithm)

Alt\Alt:	A	B	C	D
A:	0.00	0.57	0.43	0.29
B:	0.43	0.00	0.57	0.43
C:	0.57	0.43	0.00	0.57
D:	0.71	0.57	0.43	0.00

Figure 2.1 The Final Vote Matrix contains a tally of all votes received. The totals in each cell of tbe Final Vote Matrix are divided by tbe total possible number of votes to obtain tbe Average Vote Matrix.

Since a group order is a combination of multiple individual linear orderings, there may be intransitivity problems where, as in tbe above example, alternative **A** is preferred over alternative **D** by 29% of tbe group participants, **A** is preferred over **C** by 43% and **C** is preferred over **D** by 57%. Transitively, **A** would be expected to be preferred over **D** by 43% of tbe group. If **A** is preferred over **D** by 29% of the group members, then the sequence of preferences from **A** tbrougb **C** to **D** should not be larger than tbat amount.

The two top α-level sets of the above example are shown in Figure 2.2. Hagmann's algorithm maps tbis fuzzy social preference onto a nonfuzzy group ordering:

D C A B

The algorithm for this mapping is given in Appendix A, and the calculation of this mapping for this example is given in Appendix B.

R_α=.71

Alt\Alt:	A	B	C	D
A:	0	0	0	0
B:	0	0	0	0
C:	0	0	0	0
D:	1	0	0	0

R_α=.57 (all preferences at or above .57)

Alt\Alt:	A	B	C	D
A:	0	1	0	0
B:	0	0	1	0
C:	1	0	0	1
D:	1	1	0	0

Figure 2.2 Tbe matrix obtained from the α-level set where α = .71 sbows that the preference of D over A was tbe only one that had that high an α level. The matrix for α = .57 set shows many more alternative pairs baving a .57 (or above) average preference vote.

Hagmann states tbat two requirements for a Decision Support System to be adopted and to be used consistently are a) that the model be realistic and b) the results of mathematical calculations be represented in a straigbtforward manner.

The usage of a decision making tool will be encouraged when certain criteria are met. The tool nust not be cumbersome to use, because if it has a minimum of things to learn about it, new users will not be discouraged and occasional users will not feel like new users. Tbe tool should present tbe information needed without excessive verbiage that could become monotonous to frequent users. And finally, the tool should also be readily available and should not require extensive setup.

Since a user may bave several levels of use, the tool should allow the level to be chosen. For example, a user who has missed a background meeting or may be unfamiliar with tbe options being voted upon needs to read full descriptions of the alternatives. Another user who has fully investigated the options already needs only to enter a preference vote. The tool should serve botb users according to their respective needs.

The tool should allow for a limited amount of user changeability; for example, a user should be allowed to correct input after making a typing error. The tool should allow for an unlimited number of "What vote did I enter?" inquiries, and also allow a "I've changed my mind" if the voting has not been completed.

Users should be able to see how results are obtained so that the results can be verified. This will help build user confidence in the results and will thereby encourage usage of the tool.

Administration of the tool must be clear and functional. If the system presents a "secret ballot" appearance to users, then the confidentiality of the user's vote must be protected within the expected sense. Setup of a particular voting situation must not require much time. Lines of authority must be clear as to who can enter options, who can change options, who can see a particular vote history (confidentiality issue), who will be allowed to vote (security consideration), and who can determine the voting period.

The tool should allow multiple voting situations to exist on a LAN at one time. Different groups may be voting, or the same group may be voting on several separate sets of options. Each instance of a vote must have its own alternative list, alternative discussion files, voter list and vote history. If the modularity of object based programming can be maintained then each voting object can have an independence, compactness, and functionality that would be most beneficial.

The design of such a tool will be discussed in the next chapter.

CHAPTER III

PREFER TOOL DESIGN

This chapter describes the design of the PREFER tool based on the SGDSS model described in the previous chapter. The design was decomposed into four software modules which create separate processes that interact with each other over the network. Figure 3.1 shows the distribution of the modules in the LAN and Figures 3.2 through 3.7 show the data flow in this implementation.

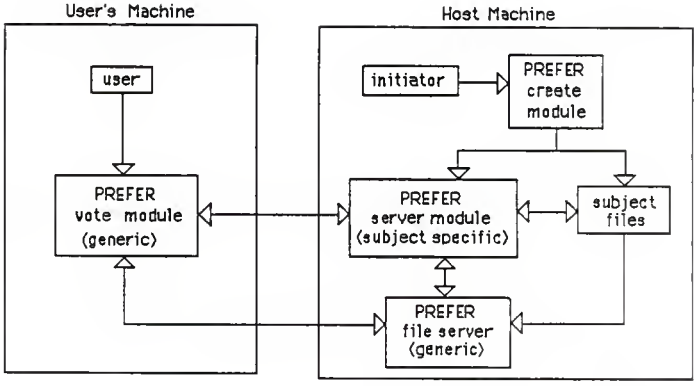

Figure 3.1 Illustration of the distribution of PREFER Tool modules among the Host computer and the User computers on a LAN

An executable voter's PREFER Interface module resides on each computer in the network. This user interface module is generic and can be used by anyone to interact with any instance of the PREFER Server. The source code for the PREFER Server module, the executable PREFER File Server module, and the executable PREFER Setup module can be hosted by any computer which is chosen to be a server.

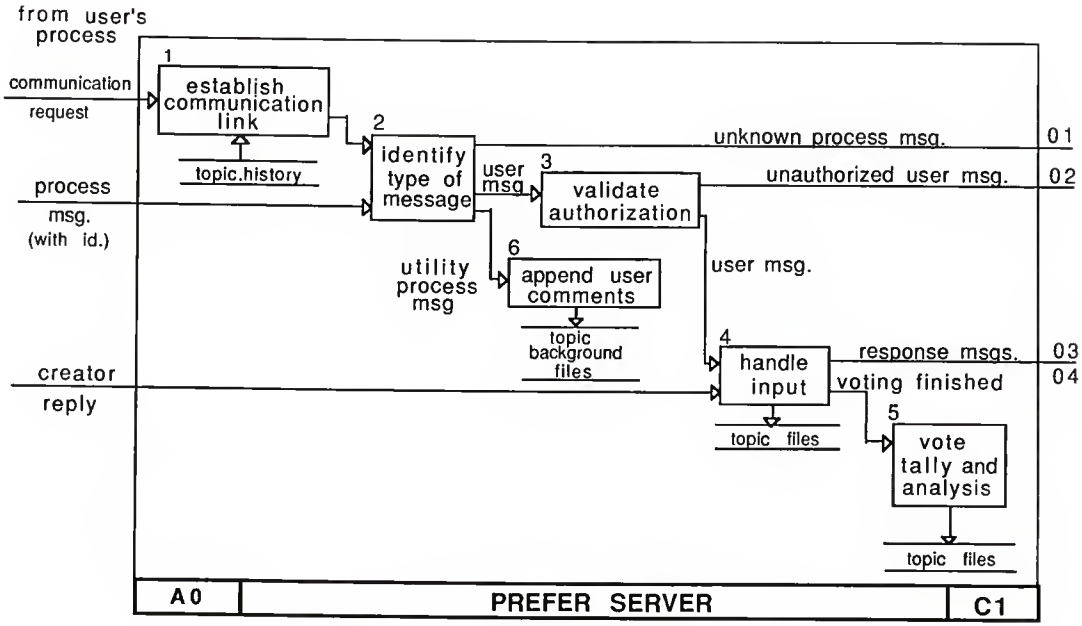

Figure 3.2 PREFER Server Module data flow diagram

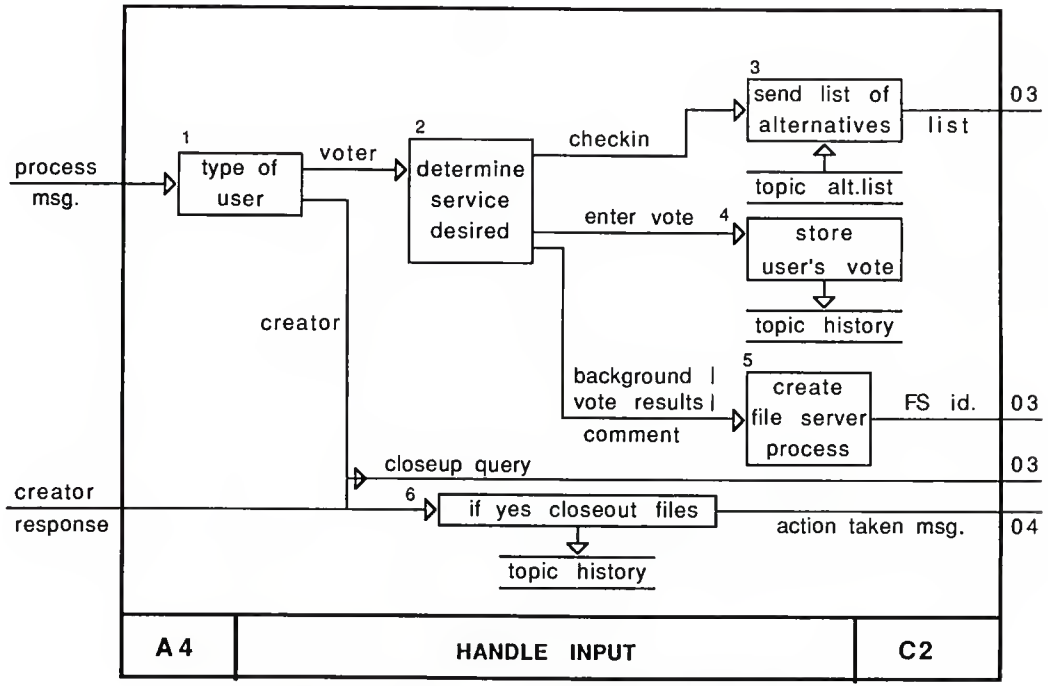

Figure 3.3 Data flow diagram of expansion of the Handle Input block of the PREFER Server data flow diagram (Fig. 3.2)

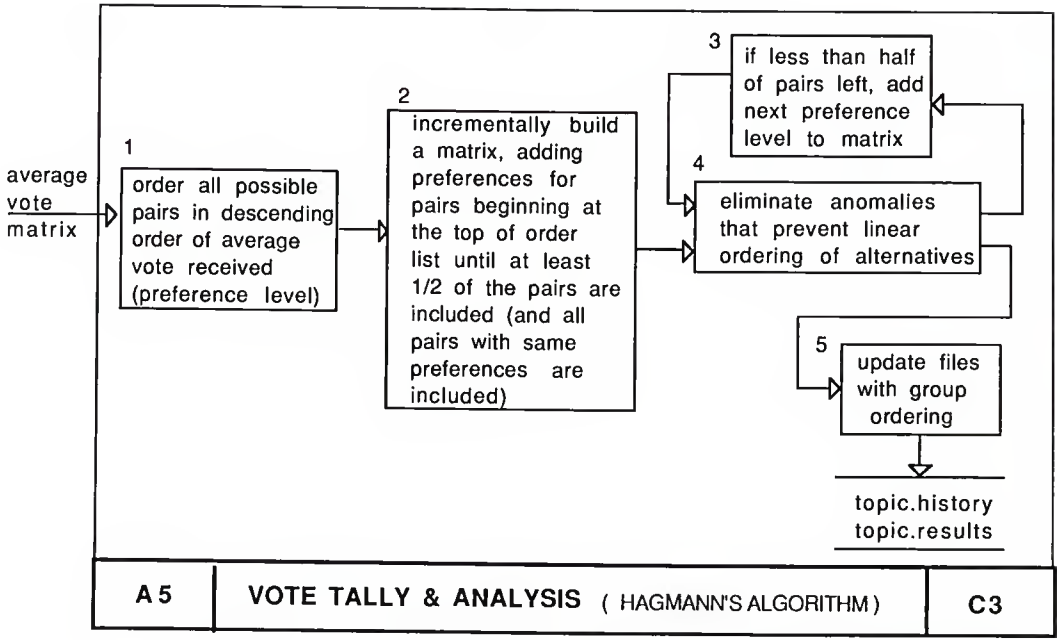

Figure 3.4 Data flow diagram of expansion of the Vote Tally and Analysis block of the PREFER Server data flow diagram (Fig. 3.2)

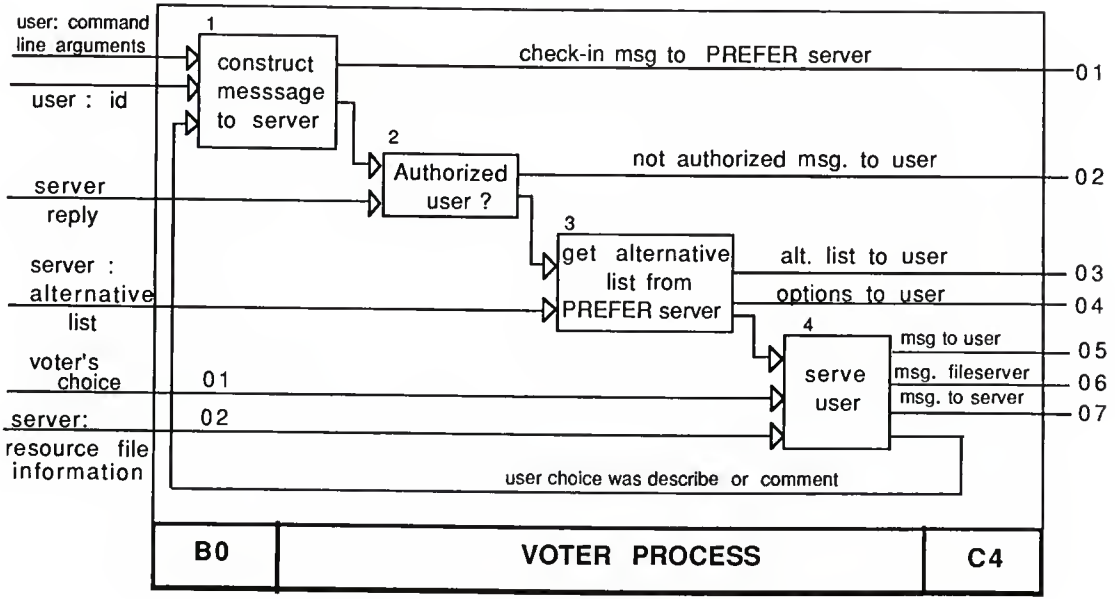

Figure 3.5 PREFER Vote Module data flow diagram

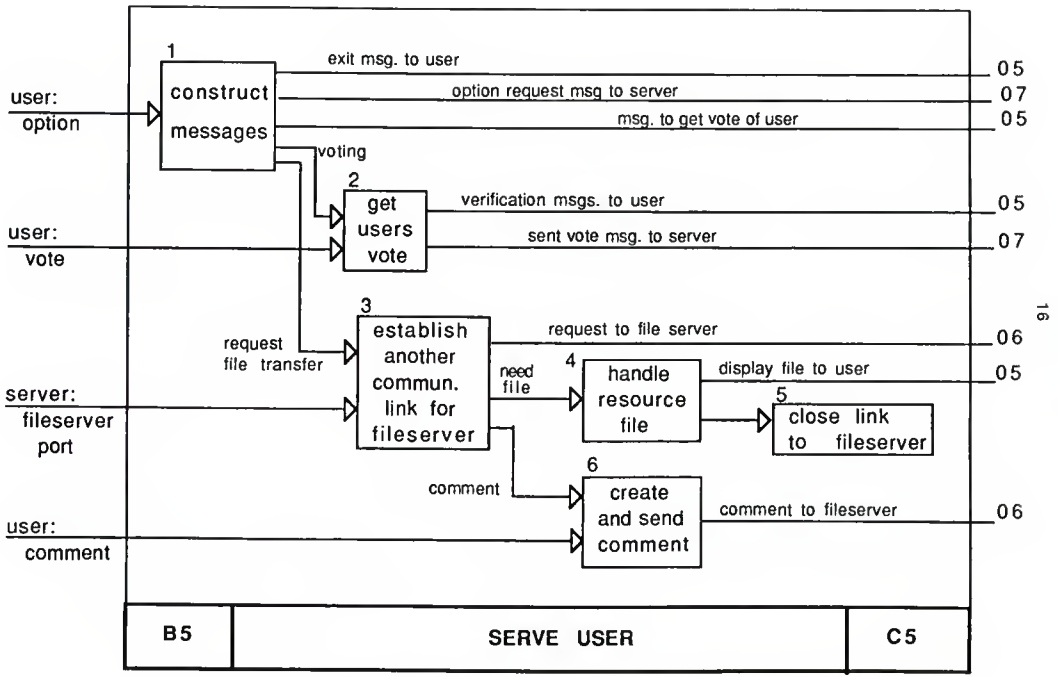

Figure 3.6 Data flow diagram of expansion of the Serve User block of the PREFER Vote User Process data flow diagram (Fig. 3.5)

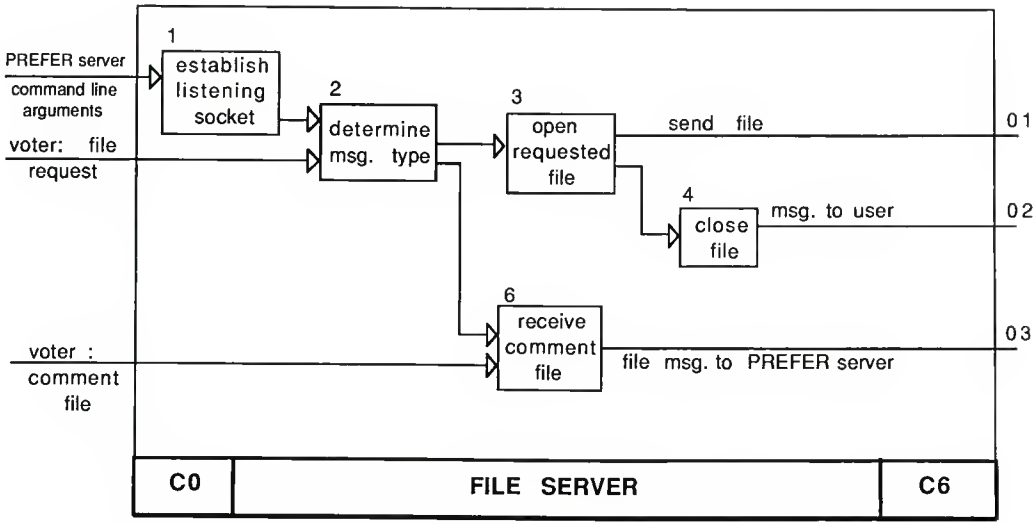

Figure 3.7 PREFER Fileserver Module data flow diagram.

An initiator who wants to create a voting session must log on the host computer and run the PREFER setup module which will prompt for information needed to set up a specific instance server module. The information needed includes topic, number of alternatives, statement of each alternative, number of authorized voters and login name of each, brief synopsis of each alternative, and final date for voting.

The PREFER setup module takes the "topic" input information and creates a server process named "topic" and the following files:

subject.h - temporarily holds specifications for instance of server
"topic".log - initially holds only a copy of this topic's subject.h file
"topic".author - holds the login name, electronic mail computer address, and an assigned key for each authorized voter
"topic".history - holds the vote history
"topic"_alt.list - the alternatives listed for the screen menu
"topic"_back.A, ... , "topic"_back.I - background files for each alternative (maximum of 9)

After the setup has been completed successfully, mail is generated to each authorized voter giving the information that is needed to reach the PREFER Server. Users are expected to extract the command line from the mail message and create their own command files. The command line for the PREFER interface module contains the following information:

PREFER topicname hostmachine socket# voter_id#

bostmachine is where the topicname server process resides
socket# is the communications link for this server on its hostmachine
voter_id# is a unique number assigned to the voter

Voting is closed when all voters have entered a vote or when the final date for voting is past. When the server process is no longer needed, the creator can remove it by using the normal voter interface. The server recognizes its creator tbrough the key number and the creator is

asked if the server should be removed. If the creator responds affirmatively, the server appends all its associated files to an archival file, "topic".log, and terminates its own operation.

There are four separate source programs:

1. "PREFER.create.c" creates all instances of the PREFER server. This program prompts the user who is creating a voting server for information needed to make the process and necessary support files. It creates all support files, including "subject.h" before initiating the compilation of "PREFER.server.c".

2. "PREFER.server.c" is generic source code which when compiled uses the specific information supplied in the "subject.h" file to create the topic voting server process.

3. "PREFER.fileserver.c" is a generic file server which acts as a transmitter of resource files to designated voters. The voting server creates a specific file server whenever a voter requests to append comments to background files or to receive information held in files on the server's computer.

4. "PREFER.vote.c", a voter's interface program, acts as the interface between voters and the vote server. Arguments given on the command line must supply the information needed to connect to the proper server process.

Source code for each of these four modules is given in Appendices E through H, respectively. Specific suggestions for installation are given in Appendix D.

Chapter four contains and explains a simulated example of PREFER as used in a specific small group decision process.

CHAPTER IV

PREFER TOOL SAMPLE USAGE

The following simulated example is developed from a voting situation in the Manhattan, Kansas school board at the time this implementation was designed. Hopefully, by examining the treatment of this topic, readers can see applications to situations familiar to them. Unfortunately, computers and computer networks are not part of the support environment available to most school boards, but perhaps some day this will change.

The school board is considering the overcrowding in several of the district school buildings. After much discussion and community input seven possible solutions are proposed and the school board must choose among them. The president of the school board initiates the voting object by typing "PREFER.create" to invoke a process which interactively leads the president through the setup procedure. A partial transcript of the setup session for this example is contained in Appendix C. Figure 4.1 on the next page shows the list as PREFER displays it on the terminals.

USD383 School Buildings:

A: Build a new school

B: Rent additional space as needed for overcrowded conditions

C: Redraw the school attendance boundaries

D: Add classroom space to existing school buildings

E: Choose specific groups of students to move out of crowded schools

F: Create split shifts to use tbe existing huildings more hours

G: Do nothing about current over-crowding at Roosevelt & Bluemont

Choose: D(descriptions), V(vote), P(pro/con comments),
or Q(quit to Op.Sys.) - (_)

Figure 4.1 Sample terminal display of alternatives and choices available to users

The mail message which each school hoard memher receives after the president creates the vote server process looks like tbis:

```
#   JDoe, you are asked to consider the following topic:
#           USD383 school buildings.
#   A PREFER server has heen set up and you are asked to
#   use the following command line to access it:

        PREFER school ksuvax1 3000 1294

#   For your convenience, please huild a executable file with
#   this line.  A suitable name for this file is "school".
#   After creating the file use,
#           cbmod 700 school
#   so that only you can use it.
```

Now when JDoe types "school", the PREFER interface process begins hy estahlishing a socket connection to the socket link number 3000 on the ksuvax1, and then prompts for JDoe's login

name. This login name and the argument key "1294" will be communicated to the PREFER "school" server on the ksuvax1. JDoe must use the key given, "1294", to be recognized as an authorized voter. If the server process detects that she is not an authorized voter for this topic, no additional information is returned and the communication link between the two processes will be broken. But if JDoe is verified to be an authorized voter, the PREFER server sends the number of alternatives, a list of the alternatives, and the current status of the vote back to her interface process.

The interface process displays this list of alternatives on JDoe's terminal screen, so she can choose among the options available. When she chooses to see background material, tbe background file on the chosen alternative is sent and stored locally; the Unix function "more" provides a flexible way for her to read such information. After "more" is exited, the file is removed from the local directory, so she is not aware tbat it resided temporarily on her machine. When finished with the background information, she again sees the original alternative list and options. She can choose to add a comment to any of the alternative background files. Comments are handled in a similar way.

If JDoe requests to vote, the following prompt appears below the menu:

Type in the alternative letters in the order of your preference.
Please use all letters A tbrough G:

Tbe process makes checks for possible errors in the input, such as duplications or unacceptable letters, and prompts the user regarding the error and asks tbe input to be repeated. Possible error prompts are:

Duplicate B entry, please reenter ordering:
Entry Z out of range, please reenter ordering:

If she has voted previously the prompt is somewhat different:

> You have already voted. Your previous vote: A B C F D E G
> Do you want to change your vote. (Y/N)

Since the following explanations would be excessively long with seven alternatives, the remaining discussion will be based upon a reduced set of only the first three alternatives.

After all voting has been completed, a different prompt appears at the end of the alternative listing as shown in Figure 4.2.

> USD383 School Buildings:
> (subset of original seven)
>
> A: Build a new school
>
> B: Rent additional space as needed for overcrowded conditions
>
> C: Redraw the school attendance boundaries
>
> The voting has been completed: group preference order - B C A
> your vote - A B C
>
> Do you want to see the group preference analysis? (Y/N)

Figure 4.2 Sample terminal display of alternatives and summary of results of completed voting

In this example, the five voters gave the following preference orderings:

voter	order
#1:	A B C
#2:	A B C
#3:	B C A
#4:	B C A
#5:	C A B

Note that in this set of preferences there is a group circular preference: A is preferred over B, B is preferred over C, and C is preferred over A.

If the group preference analysis is requested, the calculations are displayed to users as shown in Figure 4.3. This analysis file is handled in the same manner as background files.

SERVER OUTPUT
of Vote Analysis

In the vote matrix, each position shows the number of voters who prefer the row alternative over the column alternative. The total vote looks like this:

Alt\Alt:	A	B	C
A:	0	3	2
B:	2	0	4
C:	3	1	0

average vote:

Alt\Alt:	A	B	C
A:	0.00	0.60	0.40
B:	0.40	0.00	0.80
C:	0.60	0.20	0.00

OBTAIN GROUP PREFERENCE

3 non-zero matrix positions are needed to achieve
a group preference order, currently have 0 so:
Insert preference level 1.00
Insert preference level 0.80
Insert preference level 0.60

Alt\Alt:	A	B	C
A:	0.00	0.60	0.00
B:	0.00	0.00	0.80
C:	0.60	0.00	0.00

Total preferences: B > A (0.80 > 0.60)
Eliminate any preference for A over B:
since MATRIX[A][B] = 0.60, zero that position.

3 non-zero matrix positions are needed to achieve
a group preference order, currently have 2 so:
Insert preference level 0.40

Alt\Alt:	A	B	C
A:	0.00	0.00	0.40
B:	0.40	0.00	0.80
C:	0.60	0.00	0.00

Total preferences: C > A (0.60 > 0.40)
Eliminate any preference for A over C:
since MATRIX[A][C] = 0.40, zero that position.

Alt\Alt:	A	B	C
A:	0.00	0.00	0.00
B:	0.40	0.00	0.80
C:	0.60	0.00	0.00

DISPLAY RESULTS:

MATRIX array: Totals:

Alt\Alt:	A	B	C		sums	entries
B:	0.40	0.00	0.80	:	1.20	2
C:	0.60	0.00	0.00	:	0.60	1
A:	0.00	0.00	0.00	:	0.00	0

Group preference: B C A

Figure 4.3 Complete vote analysis of suggested sample vote as created by the PREFER Tool

Since renting of additional space was the most popular option from the first decision, another PREFER voting object can be set up easily with all rental possibilities being listed as the alternatives.

The ease of using voting objects should make the mechanics of their use become automatic so that attention can be focused on the decisions being made. PREFER should help the group have equal access to any research being done. Also pro and con comments can be entered without others knowing exactly who held a particular opinion. This would allow arguments for one solution or another to stand on merit alone.

The environment where PREFER was developed and tested is explained in the next chapter.

CHAPTER V

PREFER TOOL SPECIFICS & EVALUATION

SPECIFICS

PREFER was tested on a local area network at Kansas State University which included a DEC VAX 11/780, a Harris HCX-9, two AT&T 3B15's, and numerous AT&T 3B2/400's. The machines were interconnected with Ethernet hardware and TCP/IP software protocol. All of the machines were running under some version of the UNIX operating system. The program modules were written in the C programming language for ease of interfacing with the operating systems and TCP/IP. TCP/IP sockets were used for inter-process communication. Any one of the machines could have heen used as the server processor, and users could contact the server from any machine on the network.

Certain programming considerations affected the resulting programs. Since human discrimination limitations make ordering of items more and more difficult as the numher of items increases, the number of alternatives to he considered at any one time has an upper limit of nine. This limitation also facilitates the listing of all options in one screen presentation.

The list, description, and support information files must all reside on the server machine. They are not distrihuted for local machine storage to avoid duplication of support files and to facilitate the deletion of old files once a voting instance hecomes obsolete.

The creator of voting servers must know the basic information about the authorized voters: their login names and electronic mail addresses. A small amount of security is provided by the fact that each voter is given a key number which must be used when voting; however, this is not a major security check for environments where confidentiality is highly critical. A message is mailed electronically to each voter giving the necessary information on how to access the voting object. This message contains the command which the voter must use, and the message suggests that this command line be extracted from the mail message and made into an executable file. In this way the need to memorize a socket number and a key is eliminated.

Care must be taken when putting the compiled code files on the system in order to give users execution rights, but to prevent anyone and everyone from having access to the voter authorization and vote history files. The system administrator should be consulted on where to store files and what the access modes should be for both subdirectories and files. What was done for this implementation is discussed in Appendix D.

PREFER TOOL EVALUATION

This implementation meets the criteria of ease of use by prompting for user input. Each prompt indicates acceptable responses, and when an unrecognizable input is entered, a bell rings to draw the user's attention to re-enter the response. Most information displayed to the user is specific to that particular voting instance. There are no large generic menus--only brief prompts at the bottom of topic-related listings.

Server connections are hrief because all are hased on exchange of a few hytes of information. The longest exchange hetween a user and the server occurs when the alternate list is sent (no more than about 2000 hytes). The communication link is never delayed hy user response time hecause user input is done outside of connection time. Although TCP/IP protocol may queue up to five users for communication over one socket, the wait time should he insignificant for most users in most systems.

Prompts allow users to proceed straigbt to a vote or to ask for availahle background on the alternatives. When a user requests to see background material or to append a comment to hackground files, the server sets up a secondary process with a dedicated socket connection hetween the host machine and the user's machine for file transfer. The numher of sockets the host machine can establish at one time is limited, but this should not he a prohlem when hackground material is accessed by self-scbeduled users.

When a vote is completed, users see the resulting group ordering and can ask for detailed analysis of the results if there is interest. The analysis, which shows in a step-by-step fashion how the final group preference was derived, can help build confidence in users. Thus, users can utilize the PREFER tool with several levels of feedback according to need.

PREFER allows users to check a vote at several stages. It asks for a verification of the vote after it is entered to allow users to catch typing errors. Until all voting is finished, users may choose to change a previously entered vote. After all voters have voted or the voting deadline arrives, tbe server uses Hagmann's algorithm to determine the group preference if one exists. If invoked after the voting is finished, the server displays the group preference and the inquiring user's preference and asks if tbe vote analysis should he displayed. The process then

returns users to the operating system. Users can choose to return to the operating system at several points in the program.

CHAPTER VI

RESULTS AND FUTURE CONSIDERATIONS

A summary of the contributions of this implementation are given in Figure 6.1. The result was the creation of PREFER, a voting tool, whicb can be used for reaching group decisions.

* refined Hagmann's algoritbm
* designed a system that embodies the Hagmann algorithm and the goals of the SGDSS model proposed by Hagmann
* designed components tbat would allow a voting system to be distributed
* designed a voter serving "object" that can be used alone or incorporated into a more extensive implementation of Hagmann's SGDSS model
* implemented and tested tbis design in a LAN environment

Figure 6.1 Contributions made by this report and implementation

The SGDSS model looks very attractive for management decisions that need to be made by widely separated individuals who do not have the ability to hold a face-to-face meeting. If PREFER were considered for a wide area network, more attention would need to be given to security considerations. Encrypting of all transmitted messages and stored files should be considered even in a LAN where security is important.

Prefer could be used to help create a wbole group decision support system wbich can then be studied to see how such an environment affects tbe decision making process. Tbe existence of this tool can enable researchers to analyze tbe dynamics of a changed pattern of arriving at decisions by examining such questions as tbose given in Figure 6.2.

Will peer pressure take different forms?

Will members of a group find different ways to communicate their desire to influence the vote of other persons, or let their coworkers know that "I voted with you"?

Will usually quiet memhers of a group have more influence on the group decision?

Will such an environment affect the decisions that are made: are the decisions more or less risk taking?.

Will meetings become more productive?

Will long face-to-face meetings become less necessary?

Will this tool even he accepted–will managers he willing to give up any of their perceived "power" to influence others or sway a group decision?

Figure 6.2 The type of voting analysis that might be made possihle through the use of the PREFER Tool.

This is only one small tool and its use is somewhat dependent on a decision making environment like "Colah" where the whole meeting process can he aided with computer tools. For example, if after the group chooses its alternatives, it would he helpful if the voting system could analyze the impact of each alternative decision through access to organizational datahases. If the "what if" scenarios could he projected and a synopsis of the impact of that choice could he automatically generated for inclusion in the hackground information displayed, PREFER would he more valuable to the user.

Just as electronic mail has enhanced intra-office communications, perhaps PREFER will hring computer support to groups of users who need to make major decisions.

BIBLIOGRAPHY

[BLIN74] Blin, J. M. and A. B. Whinston. "Fuzzy Sets and Social Choice." Journal of Cybernetics , Vol. 3, Nunber 4, 1974.

[DEWE10] Dewey, John. How We Think , D. C. Heath & Co., Boston, 1910, pp. 68-78.

[EWBA51a] Ewbank, Henry Lee and J. Jeffery Auer. Discussion and Debate , 2nd edition, Appleton-Century-Crofts, Inc., New York, 1951.

[EWBA51b] ibid., p. 223.

[HAGM88a] Hagmann, Constanza. "An Object Oriented Design For Local Area Decision Network (LADN) Small Group Decision Support Systems", Unpublished Doctoral Dissertation, Kansas State University, 1988.

[HAGM88b] ibid., p.86.

[HAGM88c] ibid., pp. 95-100.

[STEF87] Stefik, Masrik, et al. "Beyond the Chalkboard: computer support for collaboration and Problem Solving In Meetings", Communications of the ACM , Vol. 30, Number 1, January, 1987, pp. 32-47.

APPENDIX A

Hagmann's Algorithm

1. Create 2 initial matrices:

   Matrix1 → elements in this matrix are the number of votes for each pair taken from all individual { O_h }.
   Matrix2 → elements in this matrix are the $\mu_{R_1}(a_i,a_j)$ relation components (i.e., $1/nN(\sigma_{ij})$).
   Matrix1 and Matrix2 are filled in as votes come in but group members would not be able to see them.

   When all votes are in and/or computation date and time arrives, then if no 50/50 split in voting has occurred,

2. R_α–level sets are created by sorting the $\mu_{R_1}(a_i,a_j)$ relation values in descending order.

3. Create one additional matrix:

   Matrix3 → elements in this matrix are all the binary pairs selected from the R_α sets above that have been added at every stage of the computation. This matrix, in interaction with its related total columns, helps in finding and eliminating pairs that cause intransitivity....

   Total and Entries Columns: The "Entries" column keeps count of the number of entries in each row of matrix3; the "Total" column adds up the α values for each row in matrix3.

   The computation of the linear ordering is as follows: in order to maximize

   $$\Sigma_{(a_i,a_j)\epsilon I}\mu_R(a_i,a_j),$$

   The R_α sets are examined from the highest α value to the lowest.
   Pairs at each R_α–level are included in matrix3 as follows:

4. Start with first binary pair found at highest α–level .
   4.1 Repeat until final group ordering obtained:
   Present α -level = present (a_i,a_j) pair's α -level.

   4.2 Do while present (a_i,a_j) pair's α-level = present α-level
   Place (a_i,a_j) pair in Matrix3; add 1 to entries column for α_i ; add α to Total column for a_i .

   Present (a_i,a_j) pair ← next (a_i,a_j) pair.
   enddo
   Compute summation of total number of entries in matrix3;

4.3 If total numher of entries in matrix3 ≥ (numher of alternatives * (numher of alternatives-1)/2), sort total and entries columns in descending order, according to total column figures {Note: this allows for the maximization of the total fuzzy "votes" that each alternative gets from participants}.

4.3.1 If entries column is not in strict descending order from (numher of alternatives - 1) to 0 (i.e., each row is one less than the previous one), a total strict linear order does not exist. Then,

Start with highest ranked alternative (the one with the largest number in Totals column).

Do while not last alternative.
Examine this alternative (called HR--for Highest Ranked--from here on) against next alternative down in sequence (called LR for Lower Ranked--from here on).

Do while any LR left for this alternative:
If totals for HR and LR are the same, and matrix3 shows a preference for LR over HR (LR,HR) then,

If (LR,HR) is higher than (HR,LR) then delete (HR,LR) preference pair from matrix3 and total and entries columns; re-sort;
If order changed, make the alternative that is now LR the next one to he checked against HR;
Endif
Endif
Else,
If (total for HR) > (total for LR) and there is a (LR,HR) preference pair in matrix3, then delete that preference from matrix3 and corresponding total and entries columns; re-sort.
If order changed, make the alternative that is now LR the next one to he checked against HR;
Endif
Endif
Endif
go to next LR
Enddo
Go to next HR alternative
Enddo
Endif
Endif
If summation of numher of entries in matrix3 = (numher of alternatives * (numher of alternatives-1)/2) and "number of entries" column is in strict descending order, then ordering is complete.
Endif
End repeat.

APPENDIX B

PREFER's Application of Hagmann's Algorithm

When Hagmann's algorithm is applied to the example, the process depicted below occurs:

OBTAIN GROUP PREFERENCE

6 non-zero matrix positions are needed to achieve
a group preference order, currently have 0 so:
Insert preference level 1.00
Insert preference level 0.71
Insert preference level 0.57

Alt\Alt:	A	B	C	D
A:	0.00	0.57	0.00	0.00
B:	0.00	0.00	0.57	0.00
C:	0.57	0.00	0.00	0.57
D:	0.71	0.57	0.00	0.00

Total preferences: D > C (1.29 > 1.14)
Eliminate any preference for C over D:
since MATRIX[C][D] = 0.57, zero that position.

Total preferences: C = B (0.57 = 0.57)
so check the pairings of these alternatives:
since MATRIX[B][C] > MATRIX[C][B] (0.57 > 0.00),
zero MATRIX[C][B] if not already zero.

6 non-zero matrix positions are needed to achieve
a group preference order, currently have 5 so:
Insert preference level 0.43

Alt\Alt:	A	B	C	D
A:	0.00	0.57	0.43	0.00
B:	0.43	0.00	0.57	0.43
C:	0.57	0.43	0.00	0.00
D:	0.71	0.57	0.43	0.00

Total preferences: D > B (1.71 > 1.43)
Eliminate any preference for B over D:
since MATRIX[B][D] = 0.43, zero that position.

Total preferences: B = A (1.00 = 1.00)
so check the pairings of these alternatives:
since MATRIX[A][B] > MATRIX[B][A] (0.57 > 0.43),
zero MATRIX[B][A] if not already zero.

Alt\Alt:	A	B	C	D
A:	0.00	0.57	0.43	0.00
B:	0.00	0.00	0.57	0.00
C:	0.57	0.43	0.00	0.00
D:	0.71	0.57	0.43	0.00

Total preferences: A = C (1.00 = 1.00)
so check the pairings of these alternatives:
since MATRIX[C][A] > MATRIX[A][C] (0.57 > 0.43),
zero MATRIX[A][C] if not already zero.

Alt\Alt:	A	B	C	D
A:	0.00	0.57	0.00	0.00
B:	0.00	0.00	0.57	0.00
C:	0.57	0.43	0.00	0.00
D:	0.71	0.57	0.43	0.00

Total preferences: C > B (1.00 > 0.57)
Eliminate any preference for B over C:
since MATRIX[B][C] = 0.57, zero that position.

DISPLAY RESULTS:

MATRIX array:					Totals:	
Alt\Alt:	A	B	C	D	sums	entries
D:	0.71	0.57	0.43	0.00	1.71	3
C:	0.57	0.43	0.00	0.00	1.00	2
A:	0.00	0.57	0.00	0.00	0.57	1
B:	0.00	0.00	0.00	0.00	0.00	0

Group preference: D C A B

APPENDIX C

Example Setup Using PREFER.create

The following is a partial transcript from the setup session for the school board simulated example: (user input is underlined and inserted editorial comments are in parentheses)

What is the title of this discussion topic?

School Crowding

What is a one word description of the subject? school

How many voters are there? 5

How many alternatives? 7

Enter the last day votes are allowed (YYMMDD): 881001

You will be building the "menu" that all voters will see. Each alternative should be described in one line:

School Crowding ALTERNATIVES

A: Build a new school

B: Rent additional spaceas needed for overcrowded conditions

C: Redraw the school attendance boundaries

D: Add classroom space to existing school buildings

E: Choose specific groups of students to move out of crowded schools

F: Create split shifts to use the existing buildings more hours

G: Do nothing about current over-crowding at Roosevelt & Bluemont

The following menu will be displayed:

... (the alternatives are repeated here as they will appear so that the user can verify them)

Enter the letter of any description you would like to change.
Enter zero if the menu is okay. 0

creating file school_back.A

The background file will be opened for you to insert the synopsis of the alternative (using 'vi').

Press any key to continue.

_____ (the file vi enters already contains:) _____

School Crowding Solutions:

ALTERNATIVE BACKGROUND
A: Build a new school

SYNOPSIS:

COMMENTS (pro/con):

_ _ _ _ _ (continues after user exits vi) _ _ _ _ _

creating file exam_back.B

The background file will be opened for you to insert the synopsis of the alternative (using 'vi').

Press any key to continue.

_____ (the file vi enters already contains:) _____

School Crowding Solutions:

ALTERNATIVE BACKGROUND
B: Rent a building for all sixth graders for a temporary period

SYNOPSIS:

COMMENTS (pro/con):

_ _ _ _ _ (etc. until all background files have been opened) _ _ _ _ _

You are setting up a vote server. Please enter your
login name and host computer:

Your login name: RBoss
computer name: ksuvax1

Please enter the login name and computer mail address
of everyone who is authorized to vote on this topic.

voter #1 (login name): ADall
computer name: phobos

voter #2 (login name): JDoe
computer name: harris

- - - - - - - - (etc. until all voter information has been entered)

You have entered the following information:

#	voter	e-mail addr
1	ADall	phobos
2	JDoe	harris
3	PBuck	echo
4	RBoss	ksuvax1
5	BNice	deimos

Enter the number of any voter information you would like
to change. Enter zero if the list is okay. 0

WAIT WHILE PREFER COMPILES 'school' VOTING OBJECT

ENABLE 'school' VOTING OBJECT

Socket has port #3000

Sent mail to ADall@phobos
Sent mail to JDoe@harris
Sent mail to PBuck@echo
Sent mail to RBoss@ksuvax1
Sent mail to BNice@deimos

You have created school voting object and informed the users.

APPENDIX D

Installing PREFER In a LAN with Unix Operating Systems

A directory should be established to hold all of the files related to PREFER's performance. This directory should be owned by the manager of the PREFER system, and this individual would be the only one to have direct access to the voting objects and related support files.

By placing command files in a directory that is commonly in user's .login path specification, access to PREFER can be made by simply typing the command file name. The system on the KSU network is in a personal account so users give the full pathname when invoking the command files.

The following is the command file (PREFER) which initiates a voting session:

```
#!/bin/csh
                                        # run under c shell
clear
mesg n                                  # users can't read owner's msgs
~nancy/PREFERfiles/PREFER.vote.x $1 $2 $3 $4   # topic host socket# userid
clear
mesg y                                  # turn messages back on
```

The following is the command file (PREFER.create) which initiates a voting server creation:

```
#!/bin/csh
mesg n                          # outside users can't get owner's msgs
chdir /usrb/nancy/PREFERfiles   # directory where all PREFER activity is kept
clear
PREFER.create.x
chdir /usrb/nancy
mesg y                          # turn messages back on for owner
```

Since the voting server's history and authorization files should not be available for anyone to browse, the following listings of directories show access modes that are used in this setup:

Access mode U G O	user	group	size	filename
Command Files:				
-rwxr-xr-x	nancy	nancy	325	PREFER
-rwsr-sr-x	nancy	nancy	554	PREFER.create
Directory:				
drwxrwxrwx	nancy	nancy	1536	PREFERfiles
Files:				
-rw-------	nancy	nancy	17676	PREFER.create.c
-rwsr-sr-x	nancy	nancy	34816	PREFER.create.x
-rw-------	nancy	nancy	8755	PREFER.fileserver.c
-rwx------	nancy	nancy	41984	PREFER.fileserver.x
-rw-------	nancy	nancy	39487	PREFER.server.c
-rw-------	nancy	nancy	26873	PREFER.vote.c
-rwxr-xr-x	nancy	nancy	43008	PREFER.vote.x
drwxrwxr--	nancy	nancy	512	expired
Application Files:				
-rwx------	nancy	nancy	54272	school
-rw-------	nancy	nancy	144	school.author
-rw-------	nancy	nancy	154	school.history
-rw-------	nancy	nancy	358	school.log
-rw-------	nancy	nancy	1624	school.results
-rw-------	nancy	nancy	201	school_alt.list
-rw-------	nancy	nancy	1374	school_back.A
-rw-------	nancy	nancy	1729	school_back.B
-rw-------	nancy	nancy	1289	school_back.C
-rw-------	nancy	nancy	343	subject.h

APPENDIX E

PREFER.create.c Source Program

```
/*************************************************************************
      This program was written as part of the implementation of a
      Small Group Decision Support System designed by C. Hagmann.
      It implements a distributed voting server called PREFER. The
      programming was done by Nancy Calhoun as part of a Master's
      Report project in the summer of 1988.

      This is the source file for the PREFER.create module. It is used
      to set up all instances of tbe voting server. This program prompts
      the user who is creating a server for information needed to make
      the process and necessary support files. It creates all support
      files, including "subject.h" before initiating the compilation
      of "PREFER.server.c".
 *************************************************************************/
#include <stdio.h>                      /* paths to include files vary so check */
#include <strings.h>                    /*   the requirements of your system   */
#include <ctype.h>
#include <sys/types.h>
#include <sys/time.h>                   /* the system time definitions */
#include <sys/wait.h>

/*------------- system dependent definitions ----------------------*/
#define  HARRISdir      "/u/grads/ms/nancy/"
#define  VAX_3Bdir      "/usrb/nancy/"
#define  VI             "/usr/ucb/vi"
#define  CC             "/bin/cc"

/*------------- PREFER create definitions -------------------------*/
#define  FALSE          0
#define  TRUE           1
#define  LOGINlen       9                       /* maximum lengtb of login names  */
#define  MAILADDRlen    9                       /* maximum length of email addr.  */
#define  MAXalt         9                       /* maximum number of alternatives */
#define  MAXFILENAME    256                     /* maximum filename string lengtb */
#define  OKAY           99                      /* safe number for a flag         */

typedef char line[80];                          /* define a line of 80 characters */
typedef struct verify {         int  votekey;   /* user information */
                                char user[LOGINlen];
                                char wbere[LOGINlen];
                                };
```

```
/*---------- end of PREFER create definitions ----------*/

/*************************************************************/
/*  create a flag file to prevent multiple uses of the create module
/*  at the same time.
/*************************************************************/
opening()
{
  FILE   *fd;                          /* generic file descriptor */

                    /* check to see if create session should proceed */
  if ((fd = fopen("create.busy","r")) != NULL)
  { printf("\n\007 **** PREFER.create is busy at tbis time,\n");
    printf("          please try again later. ****\n");
    fclose(fd);                        /* close existing create.busy file */
    exit(0);
  }
  else
  {  /* Make the create.busy file to prevent two create sessions at once */
    if ((fd = fopen("create.busy","w")) == NULL)
    {  perror("opening create.busy file");
       exit(5);
    }
    fprintf(fd,"The PREFER create process is currently in use.\n");
    fclose(fd);
  }
}       /* end of opening */

/*************************************************************************
 *  all purpose closing function so that create.busy will be destroyed
 *************************************************************************/
closing(type)
  int  type;
{
  int  rval;                           /* function return value   */

  if ((rval = unlink("create.busy"))< 0)       /* remove create.busy file */
    perror("in unlinking create.busy file");
  exit(type);
}       /* end of closing */

/*************************************************************************
 *  create the subject.h file that contains the specification for this
 *  particular voting object
 *************************************************************************/
header(title, subject, voters, number)
  char *title;                                 /* title of topic under discussion  */
  char *subject;                               /* one word for topic               */
  int  *voters;                                /* number of voters                 */
```

```
 int  *number;                                  /* number of alternatives          */
{
 double  deadline;                              /* last day for voting             */
 char   *filename[MAXFILENAME];                 /* structure for building filenames */
 int     flag = FALSE;
 int     i;
 FILE   *lf;                                    /* "same name" file descriptor     */
 FILE   *sf;                                    /*  subject.h file descriptor      */

 i = 0;
 system("clear");
 printf("What is the title of this discussion topic? \n   ");
 title[i++] = getchar();
 if (title[0] == '\n')                          /* discard leftover carriage return */
    i--;
 do { title[i++] = getchar();
    } while (title[i-1] != '\n');               /* tbis is tbe hard way to get a    */
 title[i-1] = '\0';                             /*   string, see below for easy way */

 do
 {
    printf("\nWhat is a one word description of the subject?  ");
    scanf("%s",subject);

                /* check to make sure this name is not already in use  */
    strcpy(filename,subject);
    strcat(filename,".log");
    if ((lf = fopen(filename,"r")) != NULL)
    { printf("\n\007 *** This word describes an existing voting object, ");
      printf("please choose another word.\n");
      fclose(lf);                               /* close existing .log file */
      flag = TRUE;
    }
    else
    {
      /*  Create the subject.h file for this topic */
      sprintf(filename,"%s", "subject.h");
      if ((sf = fopen(filename,"w")) == NULL)
      {  perror("opening subject.h file");
         closing(5);
      }
      flag = FALSE;
    }
 }  while (flag);


printf("\nHow many voters are there?  ");
scanf("%d", voters);

printf("\n How many alternatives?  ");
```

```
  scanf("%d", number);

  printf("\nEnter the last day votes are allowed (YYMMDD): ");
  scanf("%f",&deadline);

      /* Create temporary subject.h file */
  fprintf(sf,"    /* these are the particular subject definitions */\n\n");
  fprintf(sf,"#define  title  \"%s\" \n",title);
  fprintf(sf,"#define  subject \"%s\"\n",subject);
  fprintf(sf,"#define  voters  %d%25c /* number of voters */\n", *voters,' ');
  fprintf(sf,"#define  n       %d%25c /* total number of alternatives */\n",
                    *number,' ');
  fprintf(sf,"#define  deadline  %.0f%19c/* last day (YYMMDD) for voting */\n",
                      deadline, ' ');
  fclose(sf);
}   /* end of header */


/*************************************************************************
 *  creates background files for each alternative
 *************************************************************************/
backgrnd(where, subject, title, altern)
 int  where;                                  /* letter of alternative   */
 char *subject;
 char *altern;                                /* alternative description */
 char *title;
{
 FILE  *bf;                                   /* background file descriptor */
 char  *filename[MAXFILENAME];                /* structure for building filenames */
 int    rval;                                 /* function return value   */
 int    pid;                                  /* child process id        */
 union wait status;                           /* child status            */

 static char ntc[MAXalt+1] = {' ','A','B','C','D','E','F','G','H','I'};

 printf("\n\n");
 sprintf(filename,"%s%s%c",subject,"_back.",ntc[where]);

 printf("creating file  %s\n",filename);
                 /*  Create the background  file for this alternative */
 if((bf = fopen(filename,"w")) == NULL)
 {  perror("opening background file");
    closing(2);
 }
 fprintf(bf,"   %s:  ALTERNATIVE BACKGROUND\n\n", title);
 fprintf(bf,"%s\n\n", altern);
 fprintf(bf,"SYNOPSIS:\n\nCOMMENTS (pro/con):\n\n");
 fclose(bf);
 printf("The background file will be opened for you to insert tbe \n");
```

```
  printf("synopsis of the alternative (using 'vi').\n\n");
  printf("   Press any key to continue.   ");
  getchar();
  if (fork() !=0)                                  /* parent process */
  {   if ((pid = wait(&status)) == -1)
      {  perror("wait for child");
         closing(2);
      }
  }
  else                                             /* child process   */
                  /* open background file so synopsis can be entered */
      if ((rval=execl(VI,Vl,filename,0))==-1)
      {  perror("cannot start vi process \n");
         closing(2);
      }

}   /* end of backgrnd */


/*************************************************************************
 *  used to build alternative menu
 *************************************************************************/

list(subject, title, n)
  char *subject;
  char *title;
  int  n;                                          /* number of alternatives  */
{
  static char *alpha = " ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
  static char  ntc[MAXalt+1] = {' ','A','B','C','D','E','F','G','H','I'};
                                   /* number to letter conversion table */
  int    convert[256];                             /* vote conversion table   */
  int    i, j;
  line   menu[MAXalt+1];                           /* alternative menu array  */
  FILE  *mf;                                       /* menu file descriptor    */
  int    number;                                   /* menu array position     */
  char   opt;                                      /* user's input            */
  char   this_line[74];                            /* maximum input line length */
  char  *filename[MAXFILENAME];                    /* structure for building filenames */
  /*  sets up the cbaracter conversion table for vote recognition. */
  for (i=0; i<256; i++)
      convert[i] = 0;
  for (i=1,j=27; i<27; i++,j++)
  { convert[alpha[i]] = i;
    convert[alpha[j]] = i;
  };

  sprintf(menu[0],"%10c%s ALTERNATIVES \n\n", ' ',title);
```

```
printf("\n\nYou will be building the \"menu\" that all voters will see.\n");
printf("Each alternative should be described in one line:\n\n");
printf("\n%10c%s ALTERNATIVES \n", ' ',title);
for (i=1; i < n+1; i++)
{
  sprintf(menu[i]," %c:  ",ntc[i]);
  printf("\n%s",menu[i]);
  j = 0;
  this_line[j] = getchar();
  if (this_line[j] != '\n')              /* discard any leftover carriage return */
    j++;
  do {
      this_line[j++] = getchar();
    } while ( this_line[j-1] != '\n');
  this_line[j-1] = '\0';
  strcat(menu[i],this_line);
}
do { system("clear");
    printf("The following menu will be displayed:\n_____\n\n");
    for (i=1; i < n + 1; i++)
      printf("%s\n\n",menu[i]);
    printf("-----\n");
    printf("\nEnter the letter of any description you would like to change.\n");
    printf("  Enter zero if the menu is okay.  ");
    do {  opt = get_choice();
          if ( opt == '0')
            number = OKAY;
          else
          {   number = convert[opt];                  /* convert option to index */
              if ((number<1) || (number>n))           /* check for valid entry */
              {  printf("User input not valid, reenter.  ");
                 number = 0;
              }
          }
    } while (number==0);
    if (number == OKAY)                       /* create alternative menu file */
    {
      sprintf(filename,"%s%s", subject, "_alt.list");
      if ((mf = fopen(filename,"w")) == NULL)
      {  perror("opening file");
         closing(8);
      }
      fprintf(mf,"%s\n\n", menu[0]);
      for (i=1; i<n+1; i++)                       /* create background files */
      {  fprintf(mf,"%s\n", menu[i]);
         if (n<8) fprintf(mf,"\n");
         backgrnd(i, subject, title, menu[i]);
      }
      fclose(mf);
```

```
        return;
      }
      else                                  /* prompt for change in menu */
      {
        sprintf(menu[number]," %c: ",ntc[number]);
        printf("\n%s",menu[number]);
        j = 0;
        do {
            this_line[j++] = getchar();
          } while ( tbis_line[j-1] != '\n');
        this_line[j-1] = '\0';
        strcat(menu[number],this_line);
      }
    } while (opt != '0');

  printf("\n\n\nYou have completed the design of the alternative menu\n\n");
}   /* end of list */


/*************************************************************************
 *  used to get any one letter response from user
 *************************************************************************/
int get_cboice()
{ char  opt;                        /* users first input character */
  char  eat;                        /* eats rest of input line */

  opt = getchar();
  while ((opt == '\n') || isspace(opt))
    opt = getchar();
  eat = getcbar();
  while (eat != '\n')
    eat = getcbar();
  return(opt);
}   /* end of get_choice */

/*************************************************************************/
/*  tbis function gives a randomly generated number for a key to an object.  */
/*  It uses the current seconds of the computer time for the initial seed.   */
/*************************************************************************/

int  get_number()
{

  int    seed;                                  /* seed for first random number */
  struct timeval tp;                            /* structures for computer time  */
  struct tm *now;
  static int  NEW = TRUE;                       /* flag for first time through   */

  if (NEW)
  {
```

```
    gettimeofday(&tp, 0);                    /* GMT time and date          */
    now = localtime(&tp.tv_usec);
    seed = now->tm_sec;                      /* gives a variable seed      */
    srandom(seed);                           /* give seed to random # generator */
    NEW = FALSE;                             /* no longer first time through  */
  }
  return (random());
}

/*************************************************************************
 *  obtain voter name and e-mail address
 *************************************************************************/
void get_info(v,record)
  int  v;
  struct verify  *record;
{
  int  length;

  do
  {
    if (v)
      printf("\n  voter #%d (login name):  ",v);
    else
      printf("\n   Your login name:  ");   /* object creator */
    scanf("%s",record->user);
    if ((length = strlen(record->user)) > LOGINlen)
    {  length  = 0;
       printf("\007       login name is wrong length, enter again:  ");
    }
  } while  (!length);
  do
  {  printf("            computer name:  ");
    scanf("%s",record->where);
    if ((length = strlen(record->where)) > MAILADDRlen)
    {  length  = 0;
       printf("\007       mail address is wrong length, enter again:  ");
    }
  } while  (!length);
  printf("\n");
}     /* end of get_info */

/*************************************************************************
 *  create subject.author file with authorized voter information and
 *  set up the initial subject.history file (with no votes)
 *
```

```
 *  sample authorization file (5 voters):          explanation:
 *
 *       3626     nancy          ksuvax1      |     creator's information
 *       6586     ADall          phobos       |     voter identification
 *       6948      JDoe          harris       |       "        "
 *       5230     PBuck          echo         |       "        "
 *       7870     RBoss          ksuvax1      |       "        "
 *       7831     BNice          demois       |       "        "
 *      ---------------------------------------
 *                  login          e-mail
 *       id.#       name           addr.
 *
 *
 *  sample history file:                        explanantion:
 *  (5 voters, 3 alternatives)                    row    col
 *
 *   0   3626   0   1   2   3                 |    0                the creator's "row" (no vote)
 *   1   6586   1   3   1   2                 |    0     3-5        holds order position
 *   2   6948   1   1   2   3                 |   1-5               voter's rows
 *   3   5230   1   2   3   1                 |   last              the group order row
 *   4   7870   0   0   0   0                 |   0-5    1          identification numbers
 *   5   7831   1   3   1   2                 |   1-5    2          a "1" flags a stored vote
 *   0   0000   0   0   0   0                 |   1-5    3-5        stores any vote ordering
 *  ------------------------------------
 *   0    1    2   3   4   5
 *    column numbers
 *
 *
 ************************************************************************/
who_votes(subject,voters, n)
 char *subject;
 int  voters;
 int  n;
{
 char  *filename[MAXFILENAME];          /* structure for building filenames */
 int   choice;                          /* user input */
 int   i, j;
 FILE  *af;                             /* authorization file descriptor    */
 FILE  *vhf;                            /* voter history file descriptor    */
 struct verify *ids;                    /* user id. information */
 char  null_vote[3*MAXalt+1];           /* string to hold an empty vote */
 int   seed;                            /* holds newly create identification number */
 ids = (struct verify *) calloc (voters,sizeof(struct verify));
 system("clear");
 printf("      You are setting up a vote server.  Please enter your\n");
 printf("      login name and host computer:\n");
```

```
get_info(0,&ids[0]);
system("clear");
printf("Please enter the login name and  computer mail address \n");
printf("  of everyone who is authorized to vote on this topic.\n");
for (i=1; i<voters+1; i++)
    get_info(i,&ids[i]);
do
{  system("clear");
   printf("You have entered the following information:\n\n");
   printf("%8c#     voter      e-mail addr \n",' ');
   printf("%8c-     -----      ---------- \n",' ');
   for (i=1; i<voters+1; i++)
      printf("%8c%d  %8s        %s\n\n",
            ' ',i, ids[i].user, ids[i].where);
   printf("\nEnter the number of any voter information you would like \n");
   printf("   to change.  Enter zero if the list is okay.   ");

   do {
        choice = get_choice();
        choice = choice - '0';
        if ((choice < 0) || (choice > voters))
        {  printf("        invalid choice, try again:  \007");
           choice = -1;
        }
     }  while (choice == -1);
   if (choice !=0)
      get_info(choice,&ids[choice]);        /* get each voter's info */

} while  (choice != 0);

                    /* Create the authorization file for this subject */
printf(filename,"%s%s", subject, ".author");
if((af = fopen(filename,"w")) == NULL)
{  perror("opening authorization file");
   closing(6);
}
                    /* create the voter history file for this subject */
sprintf(filename,"%s%s", subject, ".history");
if((vhf = fopen(filename,"w")) == NULL)
{  perror("opening history file");
   closing(6);
}
strcpy(null_vote,"  0");                        /* hasn't voted flag */
for (i=0; i<n; i++)                  /* one for each alternative vote */
   strcat(null_vote,"  0");

for (i=0; i<voters+1; i++)
{
   do
```

```
    {                           /* assign unique id to each voter */
      seed = get_number();
      ids[i].votekey = (seed/4096)%10000;  /* get 4 digits from middle */
      for (j=1; j<i; j++)               /* check for unique number */
      { if (ids[i].votekey == ids[j].votekey)
          seed = 0;
      }
    } while (!seed);
    fprintf(af,"%4d  %9s  %s\n",
      ids[i].votekey, ids[i].user, ids[i].where);
    fprintf(vhf,"%d  %4d  %s\n", i, ids[i].votekey, null_vote);
  }
  fclose(af);
  fprintf(vhf,"%d  %d  %s\n", 0, 0, null_vote);    /* group ordering line */
  fclose(vhf);
}  /* end of who_votes */


/*****************************************************************************
 *  this module actually activates a voting object
 *****************************************************************************/
int make_object(subject,title,host,voters)

  char *subject;
  char *title;
  char *host;                                   /* network name of host computer */
  int  voters;                                  /* number of voters       */
{
  int    i;
  char  *filename[MAXFILENAME];                 /* structures for building filenames */
  char  *filename2[MAXFILENAME];
  FILE  *mf;                                    /* menu file descriptor   */
  int    pid;                                   /* child process id       */
  int    rval;                                  /* function return value  */
  int    s_locate;                              /* socket port number     */
  FILE  *sf;                                    /* subject file descriptor */
  union wait status;                            /* child status           */
  char   user[9];                               /* voter's login name     */
  int    votekey;                               /* voter's ident. number  */
  char   where[9];                              /* voter's e-mail address */

  sprintf(filename,"%s", subject);
  printf("\n\n      WAIT WHILE PREFER COMPILES '%s' VOTING OBJECT\n\n",
                                  subject);
  /*  Create the unique server file for this topic */
  if (fork() !=0)          /* parent process */
  {  if ((pid = wait(&status)) == -1)
     {  perror("wait for child");
        closing(2);
     }
```

```
}
else                /* child process:  */
                    /* compile PREFER.server.c with subject information */
{  if ((rval=execl(CC,CC,"-o",filename,"PREFER.server.c",0))==-1)
   {  perror("cannot start compilation process \n");
      closing(2);
   }
}
                                /* Enable the topic voting daemon */
if (fork() !=0)   ;                 /* parent process - does not wait */
else                /* child process - start topic server in background */
{
   if ((rval=execl(subject,subject,"&",0))==-1)
   {  perror("cannot start voting process in background\n");
      closing(2);
   }
}
printf("\n\n       ENABLE '%s' VOTING OBJECT\n\n",subject);
sprintf(filename2,"%s%s", subject, ".log");

printf("\n");                           /* funny Unix fix */
while (TRUE)       /* read socket port number as soon as it is available */
{
  if ((sf = fopen(filename2,"r")) == NULL)
    perror("waiting to get socket.port information");
  else  break;
}
fscanf(sf,"%*s %*s %*s %*c%d",&s_locate);
printf("socket port = %d\n",s_locate);
fclose(sf);
strcat(filename,".autbor");
if ((sf = fopen(filename,"r")) == NULL)
{  perror("opening authorization file");
   closing(6);
}
i = -1;             /* start below zero so creator message doesn't count */
                                /* create and mail messages to voters */
while ((rval=fscanf(sf,"%d %s %s",&votekey,user,where))!=EOF)
{
  sprintf(filename2,"%s", "tmp.mail.file");
  if ((mf = fopen(filename2,"w")) == NULL)
  { perror("opening tmp.mail.file");
    closing(6);
  }
  if (i == -1)              /* first one in file is the creator */
  { strcpy(host,where);
    fprintf(mf,"#  %s, you have just created a voting server for\n",user);
    fprintf(mf,"#       %s.\n",title);
    fprintf(mf,"#   Use tbe following command line to retire it:\n\n");
```

```
    }
    else
    {
 fprintf(mf,"#   %s, you are asked to consider the following topic:  \n",user);
 fprintf(mf,"#       %s.\n",title);
 fprintf(mf,"#   A PREFER server has been set up and you are asked to \n");
 fprintf(mf,"#   use the following command line to access it:\n\n");
    }
    if ((strcmp(where,"harris")) == 0)
       fprintf(mf," %sPREFER %s %s %d %d\n\n",
                        HARRISdir,subject,host,s_locate,votekey);
    else
       fprintf(mf," %sPREFER %s %s %d %d\n\n",
                        VAX_3Bdir,subject,host,s_locate,votekey);
 fprintf(mf,"#   For your convenience, please build a executable file with \n");
 fprintf(mf,"#   this line.  A suitable name for this file is \"%s\".",subject);
 fprintf(mf,"\n#   After creating the file use,\n");
 fprintf(mf,"#               chmod 700 %s\n",subject);
 fprintf(mf,"#   so that only you can use it.");
    fclose(mf);
    sprintf(filename,"mail %s@%s < %s", user, where, filename2);
    if ((system(filename)) == 127)
    {  perror("sending mail");
       printf("stopped on message #%d\n",i);
       closing(5);
    }
    printf("Sent mail to %s@%s\n",user,where);
    i++;                        /* i is count of voter's mail */
  }
  if (i != voters)
     printf("****  Incorrect number of messages were sent\n");
  if ((rval = unlink(filename2))< 0)
     perror("in unlinking tmp.mail.file");
  printf("\n\n\nYou bave created the '%s' voting object.\n\n", subject);
  printf("Authorized users have been notified.\n\n");
}     /* end of make_object */


/*****************************************************************************/
main()
{

  char  bost[9];                                /* host computer's network name */
  char  *filename[MAXFILENAME];                 /* structure for building filenames */
  int   n;                                      /* number of alternatives       */
  char  subject[15];                            /* subject - one word label     */
  cbar  title[80];                              /* subject title for menu list  */
  int   voters;                                 /* number of authorized voters  */

  opening();                                    /* cbeck for create.busy file */
```

```
header(title, subject, &voters, &n);                    /* CREATE:     subject.h file */
list(subject, title, n);                                        /* "topic"_alt.list file */
who_votes(subject, voters, n);                                  /* "topic".author file & */
                                                                /*  "topic".history file */
make_object(subject, title, host, voters);                      /* "topic" voting daemon */
sprintf(filename,"cat subject.h >> %s.log",subject);
system(filename);                                               /* "topic".log */

closing(0);                                                     /* destroy create.busy file */

}   /* end of main */
/*****************************************************************************/
/********* Nancy J. Calhoun, programmer ***************************************/
/********************************************** September, 1988 ************/
/*****************************************************************************/
```

APPENDIX F

PREFER.server.c Source Program

```
/*************************************************************************
    This program was written as part of the implementation of a Small Group Deci-
    sion Support System designed by C. Hagmann. It implements a distributed vot-
    ing server called PREFER. The programming was done by Nancy Calhoun as
    part of a Master's Report project in the summer of 1988.

    This is the source file for the PREFER.server module. It is used by
    PREFER.create to make the voting server daemon.
 *************************************************************************/
  /* paths to include files vary so check the requirements of your system */

#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <sys/stat.h>
#include <strings.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sgtty.h>
#include "subject.h"          /* application specific information */


/*************************************************************************/
/*---------- These are adjustable Prefer definitions ----------*/
#define  SETPORT          0        /* set to 0 if want port# to be assigned by system */
#define  DEBUG            0        /* set to 1 for debug printouts */
#define  D1               0        /* fine level debug flag       */
/*---------- PREFER server definitions ----------*/
#define  TRUE             1
#define  FALSE            0
#define  DOWN             1
#define  UP               0
/*---------- sizing definitions ----------*/
#define  BUFFERSIZE       1024     /* temporary storage buffer      */
#define  IDlen            4        /* size of identification number */
#define  LOGINlen         9        /* maximum length of login name  */
#define  MAXalt           9        /* maximum number of alternatives */
```

```
#define  MAXFILENAME      256        /* maximum length of a filename   */
#define  MAXVSIZE         30         /* space for status information & previous vote */
#define  N_pairs          n*n        /* n is defined in the include file subject.h */
/*------------------------- action codes ------------------------------*/
#define  LEAVE            0          /* wants to exit voter process */
#define  ENTER_VOTE       1          /* wants to enter a vote       */
#define  BACKGROUND       2          /* wants to see a background file */
#define  GET_RESULTS      3          /* wants to see the results file  */
#define  CHECKIN          4          /* msg contains voter ident.   */
#define  COMMENT          5          /* wants to attacb a comment   */
/*------------------------- type definitions --------------------------*/
typedef double              array[n+1][n+1];
typedef char                buffer[BUFFERSIZE];
typedef double              narrow_array[n+1][4];
typedef struct verify {     int  votekey;              /* structure of authorization */
                            char user[LOGINlen];
                            char where[LOGINlen];
                            };
/*------------------------- end of definitions ------------------------*/

int    convert[256];                    /* global vote conversion table */

/*****************************************************************************
 *    blanks out the voting array
 *****************************************************************************/
void  blankvote (O,V)
  int   O[n+1];
  array V;
{  int  i,j;                            /* index variables */
  printf("\n");
  for (i=0; i < n+1; i++ )
  {   for ( j = 1; j < n+1; j++)
        V[i][j] = 0;
      O[i] = 0;
  }
}   /* end blankvote */

/*****************************************************************************
*  establisb a communication link - a "listening socket"
*     this is an interface to the transport level protocol (TCP/IP)
*****************************************************************************/
int socket_setup()
{

    char *filename[MAXFILENAME];        /* structure to hold filenames   */
    int    length;
    FILE   *sf;                         /* "subject.log" file descriptor */
```

```
int    sock;                              /* socket descriptor          */
struct sockaddr_in server:                /* holds socket information   */


                /* Create socket - an endpoint for communication */
sock = socket(AF_INET, SOCK_STREAM, 0);
                        /* AF_INET - ARPA internet addresses */
                        /* SOCK_STREAM - sequenced 2-way     */
                        /*     connection based byte streams */
if (sock < 0)
{  perror("server:  opening stream socket");
   exit(1);
}

server.sin_family = AF_INET;            /* Name socket using wildcards */
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(SETPORT);
      /* the SETPORT number must be converted to network byte order so
       * that the proper port will be requested.  If SETPORT is zero the
       * system assigns a port number.
       */

length = sizeof(server);
if (bind(sock, &server, length))        /* binds a name to the socket */
{  perror("server:  binding stream socket");
   exit(1);
}
/* Find out assigned port number and print it out */
if (getsockname(sock, &server, &length))
{  perror("server:  getting socket name");
   exit(1);
}
sprintf(filename,"%s.log",subject);    /* socket port # is put in file */
if ( (sf = fopen(filename,"w")) == NULL )
{  perror ("server:  *** file for socket info. could not be opened. \n");
   exit(1);
}
fprintf(sf,"Socket has port #%d\n", ntohs(server.sin_port));
fclose(sf);
printf("Socket has port #%d\n", ntohs(server.sin_port));

/* Start accepting connections */
listen(sock, 5);

                                        /*  release terminal initiating this daemon */
ioctl(0,TIOCNOTTY,0);                   /* system call so daemon will not */
                                        /*  continue to tie up a terminal */
if (DEBUG)                              /* diagnostic print */
   printf("server: Socket has port #%d\n", ntohs(server.sin_port));
else
```

```
    {  fclose(stdin);
       fclose(stdout);
       fclose(stderr);
    }
    return(sock);
}   /* end of socket setup */

/*****************************************************************************
 *  this bubble sorts either XX or TE array into nonascending order on given
 *  column.  The array named is sorted, and since the only difference in the
 *  the two arrays, is tbe size, the size must be given as a paramater.
 *****************************************************************************/
void sort(name, size, column, direction)

 double  name[N_pairs][3+1];          /* the maximum array is declared but  */
 int     size;                        /* this is the size of array named    */
 int     column;                      /* sort array on this column          */
 int     direction;                   /* sort order-ascending or descending */
{
 int     count;                       /* counts switches for debugging info.*/
 int     i,j,k;                       /* index variables                    */
 double  temp[3+1];                   /* temporary storage during switch    */
 int     which;                       /* results of comparsion for sort     */


 count = 0;
 if (D1)                        /* fine detail diagnostic printing    */
 { printf("before sort\n");
   for (i=1; i < size+1; i++)
     printf("%.2f  %.2f  %.2f\n", name[i][1], name[i][2], name[i][3]);
   printf("\n\n");
 }

 for (k=1; k < size; k++)
   for (i = k; i > 0; i--)
   {  count++;
      whicb = (direction < DOWN ? (name[i][column] > name[i+1][column])
                                : (name[i][column] < name[i+1][column]));
      if (which)
        for (j=1; j<3+1; j++)
        { temp[j] = name[i][j];                          /* switch */
          name[i][j] = name[i+1][j];
          name[i+1][j] = temp[j];
        }
      else i = 0;
   }
 if (D1)                        /* fine detail diagnostic printing */
 {
   printf(" count = %d\n",count);
   printf("after sort\n");
```

```
    for (i=1; i < size+1; i++)
      printf("%.2f  %.2f  %.2f\n", name[i][1], name[i][2], name[i][3]);
    printf("\n\n");
  }

}  /* end of function sort */

/*************************************************************************
 *  check to see if the TE array show a strict linear ordering of alternatives
 *************************************************************************/
int ordered(TE)
  narrow_array TE;
{
  int   checkflag, i;

  i = 1;
  checkflag = TRUE;
  if (TE[1][3] != n-1)                  /* top item must have n-1 elements */
    checkflag = FALSE;                  /*    for ordering to be complete  */
  while (checkflag && (i < n))
  {                                     /* fine detail diagnostic printing */
    if (D1) printf("TE[%d][3] = %.2f,  TE[%d][3] = %.2f\n",
                    i,    TE[i][3],     (i+1), TE[i+1][3]);
    if (TE[i][3] == TE[i+1][3] + 1)
      i++;
    else
      checkflag = FALSE;
  }
  return(checkflag);
}    /*  end function ordered  */

/*************************************************************************
 *   rearranges MATRIX array for final output to show group preference
 *************************************************************************/
int  resort(MATRIX,T)
 array MATRIX;
 narrow_array  T;
{
 int  i, j, k;                                  /* looping variables */
 double   *X;
       /* Note: Must be dimensioned to hold largest -- (n+2) or (voters+1) */

 int  dimen;                                /* for holding largest dimension */

 dimen = ((n+2) < (voters+1) ? (voters+1) : (n+2));
 X = (double *) malloc ( dimen*sizeof(double));
 for (i = 1; i < n; i++)
   for (j = i +1; j < n+1; j++)
     if (T[i][1] < T[j][1])                     /* exchange the rows */
```

```
    {  for (k = 0; k < n+1; k++)
       {  X[k] = MATRIX[i][k];
          if (D1)
             printf("MATRIX[%d][%d] = %.2f  then  ", i, k,MATRIX[i][k] );
          MATRIX[i][k] = MATRIX[j][k];
          MATRIX[j][k] = X[k];
          if (D1)  printf("%.2f\n",MATRIX[i][k] );
       }
       for (k = 1; k < 3; k++)
       {                              /* prints are for debugging only */
          X[k] = T[i][k];
          if (D1)  printf("T[%d][%d] = %.2f  then  ",i, k,T[i][k] );
          T[i][k] = T[j][k];
          T[j][k] = X[k];
          if (D1)  printf("%.2f\n", T[i][k] );
       }
       if (D1)  printf (" i = %d\n",i);
    }
}          /* end function resort  */

/*****************************************************************************
 *   generic array printout
 *****************************************************************************/
void  writearray (where,arrayname,type,T)

 FILE   *where;                        /* where output will be directed */
 array  arrayname;
 int    type;              /* type:    0, integer format,
                                       1, floating point format,
                                       2, floating pt. and add totals */
 narrow_array T;                       /* totals */
{
 int  i,j;                             /* index variables  */
 static char ntc[MAXalt+1] = {' ','A','B','C','D','E','F','G','H','I'};

 fprintf(where,"\n");
 if (type == 2)                                /* print title line */
 {  fprintf(where,"MATRIX array: ");
    for (i=1;i<n;i++)
       fprintf(where,"%s","    ");
    fprintf(where,"Totals:\n\n");
 }
 for ( i=0; i < n+1; i++ )
 {  for ( j=0; j < n+1; j++ )
    {
       if (i == 0)
          if (j == 0)  fprintf(where,"Alt\Alt: ");
          else
          {  fprintf(where,"%c   ", ntc[j]);
```

```
          if (type != 0)
            fprintf(where,"  ");
        }
      else
        if (j==0)
        {  fprintf(where,"  %c:   ",ntc[(int)arrayname[i][j]]);
          if (type == 0)
            fprintf(where,"  ");
        }
        else if (type==0)
                fprintf(where,"%.0f   ", arrayname[i][j]);
              else
                fprintf(where,"%.2f  ", arrayname[i][j]);
    }
    if (type == 2)
    {  if (i==0)
        fprintf(where," sums  entries");
      else
      {  fprintf(where,":  ");
        fprintf(where,"%.2f  %.0f", T[i][1], T[i][2]);
      }
    }
  fprintf(where,"\n\n");
  }
}   /* end of function writearray */

/*****************************************************************************
 *  read in past history
 *****************************************************************************/
int get_history(H)
  int  H[voters+2][n+3];                    /* history matrix */
{
  char *filename[MAXFILENAME];              /* structure to hold filenames */
  FILE *h;                                  /* history file descriptor     */
  int  i, j;                                /* index variables             */
  int  voted;                               /* number of votes in history  */

  strcpy(filename, subject);
  strcat(filename,".history");
  if ( (h = fopen(filename,"r")) == NULL )
  {  perror ("server:  *** history file could not be opened. \n");
    exit(1);
  }
  else
  {  voted = 0;
     for (i=0; i < voters+2; i++)
     {  for (j=0; j < n+3; j++)
           fscanf(h,"%d  ", &H[i][j]);
```

```
        voted += H[i][2];       /* H[row][2] is 1 if user has voted */
      }
    }
    fclose(h);
    return(voted);
}    /* end of get_history() */

/*************************************************************************
 *  update history file
 *************************************************************************/
void write_history(H)
    int  H[voters+2][n+3];                  /* history matrix  */
{
    FILE *h;
    char *filename[MAXFILENAME];            /* structure to hold filenames */
    int  i, j;                              /* index variables */

    strcpy(filename, subject);
    strcat(filename,".history");
    if ( (h = fopen(filename,"w")) == NULL )
    {  perror("server:  ** History file could not be opened for updating.\n");
       exit(1);
    }
    for (i = 0; i < voters +2; i++)
    { for (j=0; j < n+3 ; j++)
        if (j==1)
          fprintf(h,"%4d  ", H[i][j]);
        else
          fprintf(h,"%d  ", H[i][j]);
      fprintf(h,"\n");
    }
    fclose(h);
}    /* end of write_history */

/*************************************************************************
 * initializes the arrays with zeros and alternative numbers
 *************************************************************************/
void blankarray(arrayname)
    array  arrayname;
{  int  i, j;                           /* index variables        */

    for (i=0; i < n+1; i++ )
    {
      for ( j = 0; j < n+1; j++)                /* fill arrays with zeros */
        arrayname[i][j] = 0;
      arrayname[i][0] = i;          /* store alternative number in column 0 */
    }
}  /* end of blankarray */
```

```
/*************************************************************************
 * Hagmann's algorithm obtains a group preference from individual
 *      preference orderings:
 *************************************************************************/
void algorithm(M2,results)

  array  M2;                              /* matrix holding "average" votes       */
  FILE *results;                          /* vote analysis file descriptor        */
{
  int    H[voters+2][n+3];                /*  history array      */
  array  MATRIX;                          /*  working matrix     */
  narrow_array T;                         /*  matrix row totals  */
  narrow_array TE;                        /*  column 1 contains alternative 1 .. n   */
                                          /*  column 2 contains total of preferences */
                                          /*  column 3 - # of preferences for alt. n */
  double XX[N_pairs+1][3+1];              /*  column 1 & 2 represent (a1,a2) pairs   */
                                          /*  column 3 - alpha associated with pair  */
  double RLEVEL;                          /* alpha level being examined */
  double ALPHA;                           /* alpha value for pairs      */
  int    I=0, J=0, K=0, M=0, P=0;         /* index variables            */
  int    SUMENTRIES = 0;                  /* number of pairs entered in matrix */
  double CHECK1;                          /* flags for checking for any */
  double CHECK2;                          /*    change after adjustment */
  int    i,j;                             /* loop variables             */
  int    count;                           /* used to generate pairings  */
  int    COMPLETEFLAG = 0;                /* set when ordering achieved */
  int    DOCHECK = FALSE;                 /* flag                       */
  int    temp;                            /* used to avoid repeat       */
  double dtemp;                           /*         indexed lookups    */
  int    a1, a2;                          /* alternative (a1,a2) in pair */

  static char ntc[MAXalt+1] = {' ','A','B','C','D','E','F','G','H','I'};

  blankarray(MATRIX);                      /* blank all arrays used */
  for (i=0; i < n+1; i++ )
  { for (j=0; j < 4; j++)
    {  TE[i][j] = 0;
       T[i][j] = 0;
    }

    TE[i][1] = i;                         /* column 1 holds alternative # */
    T[i][0] = i;                          /* column 0 holds alternative # */
  }
  count = 0;
  for (i=1; i < n+1; i++)                 /* array of all possible pairs (a1,a2) */
  {  for (j=1; j < n+1; j++)              /*  col: 1   2      3                  */
     {                                    /*       a1  a2   avg preference vote  */
       count++;
       XX[count][0] = count;              /* pair row */
       XX[count][1] = i;                  /*  a1      */
```

```
        XX[count][2] = j;                    /*  a2      */
        XX[count][3] = M2[i][j];             /*  a1 preferred over a2 vote */
      }
    }
    sort(XX, N_pairs, 3, DOWN);          /* sort hy decreasing pair vote */
     /*  set variahles to hegin at alpha level 1 and at the first of array  */
     RLEVEL = 1;
     K = 1;
     I = XX[K][1];
     J = XX[K][2];
     ALPHA = XX[K][3];
     if (DEBUG) printf("I = %d, J = %d, ALPHA = %.2f\n",I,J,ALPHA);
     COMPLETEFLAG = FALSE;
     fprintf(results, "\n        OBTAIN GROUP PREFERENCE \n\n");

   /* loop so long as all pairs have not heen examined, alpha level has not    */
   /*      reached zero, and the completeflag has not been set to true.        */

     while ((K <= N_pairs) && (ALPHA > 0) && (!COMPLETEFLAG))
     {
  /* inner loop 1 do as long as have not reached last pair, the number of pairs */
  /*   represented hy alpha levels in the matrix is less than n*(n-1)/2, and   */
  /*   the alpha level is greater than zero.                                    */

       fprintf(results,"%d non-zero matrix positions are needed to achieve\n",
                          n*(n-1)/2);
       fprintf(results," a group preference order, currently have %d so:\n",
                                              SUMENTRIES);

       while ((K < N_pairs) && (!DOCHECK) && (ALPHA > 0))

  /* inner loop 2, do as long as this pair's alpha level is the same as the    */
  /*   level we are inspecting, insert the alpha value into the matrix and     */
  /*   add this value into the sum of all alpha values for the preferred       */
  /*   alternative.  Also increment the count of the entries for this alter-   */
  /*   native and the count for the total numher of pairs examined.            */
       {
         sort(TE, n, 1, UP);
         while ( ALPHA == RLEVEL )
         {
           MATRIX[I][J] = ALPHA;
           TE[I][2] = TE[I][2] + ALPHA;
           TE[I][3]++;
           SUMENTRIES++;

   /* examine the next pair */
           K++;
           I = XX[K][1];
           J = XX[K][2];
```

```
        ALPHA = XX[K][3];
      }
/* a new alpha value has been encountered so set the examination level to it */
      fprintf(results,"Insert preference level %.2f\n",RLEVEL);
      RLEVEL = ALPHA;
      if (SUMENTRIES >= n*(n-1)/2)
        DOCHECK = TRUE;
    } /* end while K<N_pairs and !DOCHECK, and ALPHA>0 */

    writearray(results,MATRIX,1,0);

/* if the numbers of pairs examined has reached this value, enough pairs have */
/*    been examined to have a possible ordering.                               */
    if (DOCHECK)
    {
     sort(TE, n, 2, DOWN);
     if ((SUMENTRIES == n*(n-1)/2) && (ordered(TE)) )
       COMPLETEFLAG = TRUE;
     else
     {
       M = 0;
       P = 0;
/*  compare alternate M with all alternatives at equal or lower overall alpha
 *   levels to find anomalies where the alternative pairings show  reversing
 *   of preference
 */
       while (M < n-1)
       {  M++;
          P = M;
          while (P < n)
          {  P++;
             CHECK1 = TE[P][1];
             CHECK2 = TE[M][1];
             a1 = TE[M][1];
             a2 = TE[P][1];
             if (DEBUG)  printf("compare %d : %d\n", a1, a2);
/*  if overall preference is the same and alternative P preferred over M,
 *   drop vote for M over P */
             if (TE[M][2] == TE[P][2])
             { if (MATRIX[a2][a1] > MATRIX[a1][a2])
               {
   fprintf(results,"Total preferences:  %c = %c  (%.2f = %.2f)\n",
                              ntc[a1],ntc[a2],TE[M][2],TE[P][2]);
   fprintf(results,"so check the pairings of these alternatives:\n");
   fprintf(results," since MATRIX[%c][%c] > MATRIX[%c][%c] (%.2f > %.2f),\n",
           ntc[a2],ntc[a1],ntc[a1],ntc[a2],MATRIX[a2][a1],MATRIX[a1][a2]);
   fprintf(results,"    zero the MATRIX[%c][%c] position.\n",ntc[a1],ntc[a2]);

               if ( MATRIX[a1][a2] !=0)
```

```
                { TE[M][2] = TE[M][2] - MATRIX[a1][a2];
                  MATRIX[a1][a2] = 0;
                  TE[M][3]--;
                  SUMENTRIES--;
                }
                sort(TE, n, 2, DOWN);    /* readjust ordering */
              }
            }
            else
/* if overall M alternative is preferred over P, drop any vote for P over M */
              if (TE[M][2] > TE[P][2])
                if ( MATRIX[a2][a1] > 0)
                {
  fprintf(results,"Total preferences:  %c > %c   (%.2f > %.2f)\n",
                          ntc[a1],ntc[a2],TE[M][2],TE[P][2]);
  fprintf(results,"Eliminate any preference for %c over %c:\n",
                                  ntc[a2], ntc[a1]);
  fprintf(results,"  since MATRIX[%c][%c] = %.2f, zero that position.\n\n",
                        ntc[a2],ntc[a1],MATRIX[a2][a1]);

                  TE[P][2] = TE[P][2] - MATRIX[a2][a1];
                  MATRIX[a2][a1] = 0;
                  TE[P][3]--;
                  SUMENTRIES--;
                  sort(TE, n, 2, DOWN);        /* readjust ordering */
                }
            if (CHECK1 != TE[P][1])
            { P--;                /* change so recheck same position */
              writearray(results,MATRIX,1,0);
            }
            if (CHECK2 != TE[M][1])
            { P = M;              /* change so recheck against ALL   */
                                  /*     lower positions             */
              writearray(results,MATRIX,1,0);
            }
        }   /*  end while P < n  */
      } /*  end while M < n-1  */
      if ((SUMENTRIES == n*(n-1)/2) && (ordered(TE)) )
        COMPLETEFLAG = TRUE;
      else
      {
        sort(TE, n, 1, UP);
        DOCHECK = FALSE;
      }
    }    /*  end else if SUMENTRIES == .. & ordered */
  }      /*  end enough SUMENTRIES & ALPHA > 0  */
}        /* while (K <= N_pairs && ALPHA > 0 && !COMPLETEFLAG)  */
get_history(H);
H[voters+1][0] = 1;    /* shows group preference has been calculated */
```

```
    if (COMPLETEFLAG == TRUE)
    {
     fprintf(results,"\nDISPLAY RESULTS:\n");

/*  print out final array arranged in group preference order with details   */
      for (i = 1; i < n+1; i++)
       for (j = 1; j < n+1; j++)
       {
        dtemp = MATRIX[i][j];  /* use dtemp to avoid extra index lookup */
        if (dtemp > 0)
        {
         T[i][1] = T[i][1] + dtemp;
         T[i][2]++;
        }
       }
      resort(MATRIX,T);
      writearray(results,MATRIX,2,T);
      fprintf(results,"\n\n");
      fprintf(results,"Group preference: ");     /* print to results file */
      for (i = 1; i < n+1; i++)
      {
        temp = TE[i][1];
        H[voters+1][i+2] = temp;
        fprintf(results,"%c ", ntc[temp]);
      }
      fprintf(results,"\n\n");
      if (D1)  for (i =1; i<voters+2; i++)
                 {  for (j=0;j<n+3;j++)
                      fprintf(results,"%d ",H[i][j]);
                    fprintf(results,"\n\n");
                 }
    }
    else
    { fprintf(results," ***  STRICT LINEAR ORDERING NOT POSSIBLE.\n\n");
      if (DEBUG)  printf(" ***  STRICT LINEAR ORDERING NOT POSSIBLE.\n\n");
    }
    fprintf(results,"\f\n\n");
    fclose(results);
    write_history(H);          /* update history file with group ordering */
}  /* end of algorithm */

/*************************************************************************
 *  create an instance of a file server to take care of this request
 *************************************************************************/
void create_fs(msgsock)
 int  msgsock;
{
 buffer buf;                              /* input buffer */
 char  *filename[MAXFILENAME];            /* structure to hold filenames  */
```

```
FILE  *fs;                                  /* socket info. file descriptor */
int    pid;                                 /* child id                     */
int    rval;                                /* function return value        */
struct stat  file_status;                   /* socket info. file status     */
union wait status;                          /* child process return status  */

if (DEBUG)  printf("user wants to get resource file\n");

if (fork() !=0)                                 /* parent process */
{   if ((pid = wait(&status)) == -1)
    {  perror("server:  wait for child");
       exit(2);
    }
}
else                        /* child process - create a fileserver process */
if ((rval=execl("PREFER.fileserver.x","PREFER.fileserver.x",subject,0)) == -1)
{  write(1, "cannot start second process \n",34);
   exit(2);
}
/* After being created the fileserver writes its socket id. into a file.  The */
/* voting server must read tbis file to get the information for the voter.    */

strcpy(filename,"prefer.fs");
do { fs = fopen(filename,"r");              /* open socket information file */
   } while (fs == NULL);
do {                                        /* anything in file yet? */
     if ((rval = stat(filename,&file_status)) < 0)
     {  perror("server:  file status");
        exit(1);
     }
   } while (file_status.st_size == 0);          /* loop until fs has length */

bzero(buf, sizeof(buf));
if ((rval = fread(buf, sizeof(cbar), 1023, fs)) < 0)   /* read into buffer */
{  perror("server:  reading socket id");
   exit(1);
}
                        /* send fileserver socket information to voter */
if (write(msgsock, buf, strlen(buf)) < 0)
{  perror("server: writing on stream socket");
   exit(1);
}
fclose(fs);
if ((rval = unlink(filename))< 0)           /* remove temporary info. file */
{  perror("in unlinking prefer.fs");
   exit(1);
}
}  /*  end of create_fs */

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

```
* this is an authorized user who can vote
*    server must send to authorized user:    # of alternatives,
*                                            if votes are completed,
*                                            if this user has voted,
*                                            user's previous vote
*                                            group vote
*****************************************************************************/
void serve_user(what,voted,msgsock,H,row,thisvote,day)
 int  what;                                  /* what authorized user wants to do */
 int  *voted;                                /* voted can be changed within this module */
 int  msgsock;
 int  H[voters+2][n+3];                      /* bistory matrix */
 int  row;
 char *thisvote;
 double  day;
{
 buffer buf;                                 /* message buffer    */
 char  *filename[MAXFILENAME];               /* structure for building filenames   */
 int    finished = FALSE;                    /* set to 1 when voting is finished   */
 int    i, j;                                /* index variables                    */
 FILE  *list;                                /* list of the subjects alternatives  */
 int    listflag;                            /* signals completion of reading file */
 char  *order;                               /* used to bold vote                  */

 static char  convert_vtc[10] = { '@','A','B','C','D','E','F','G','H','I' };

 if ((*voted == voters) || (day > deadline))
   finisbed = TRUE;

 if (wbat == CHECKIN)
 {
   order = (char *)malloc((4*n+2)*sizeof(char));
   for (i=1; i< 4*n+2; i++)                  /* blank new allocated memory */
     order[i] = ' ';

             /* convert to output string array that lists vote stored */
   for ( i = 1; i < n+1; i++ )
   {
     order[i+(i-1)] = convert_vtc[H[row][i+2]];
     j = i + n + 1;
     order[j+(j-1)] = convert_vtc[H[voters+1][i+2]];
   }
   order[2*n+1] = '\0';                      /* end of string */
   order[4*n+2] = '\0';                      /* end of string */

   if ((H[voters+1][0]==1) && (H[voters+1][3]==0))
     sprintf(buf,"%d %d %d %s %s", n,finished, H[row][2],
                 &order[1],"COULD NOT ORDER");
   else
```

```
    sprintf(buf,"%d %d %d %s %s",
            n,finished, H[row][2],&order[1],&order[2*n+3]);
  if (write(msgsock, buf, 1024) < 0)          /* send info. to user */
  {  perror("server:  writing on stream socket");
     exit(1);
  }
  strcpy(filename, subject);          /* setup to send alternative list */
  strcat(filename,"_alt.list");
  if ( (list = fopen(filename,"r")) == NULL )
  {  perror ("S: *** alternative list file could not be opened. \n");
     exit(1);
  }
  else
  { listflag = TRUE;
    while (listflag)                                  /* read list */
    {
      bzero(buf, sizeof(buf));
      if ((listflag = fread(&buf[1], sizeof(char), 1023, list)) < 0)
      {  perror("server:  reading list file");
         exit(1);
      }
      else if (listflag < 1023)
                          /* tell user another read is unnecessary */
          {  listflag = FALSE;
             buf[0] = '0';
          }
          else  buf[0] = '1';
      if (write(msgsock, buf, 1024) < 0)          /* send menu list msg */
      {  perror("server:  writing on stream socket");
         exit(1);
      }
    }

    fclose(list);
  }
}   /* end of checking in */
else
  if (what == ENTER_VOTE)                          /* user has sent a vote */
  {                                  /* store it in history array */
    if (H[row][2] == 0)
    {  H[row][2] = 1;
       (*voted)++;
    }
    for (i=0; i<n; i++)
       H[row][i+3] = convert[thisvote[i]];

    write_history(H);
  }
  else                                    /* fileserver is needed */
```

```
    if ((what == BACKGROUND) || (what == GET_RESULTS) || (what == COMMENT))
      create_fs(msgsock);

}       /* end serve_user - the user has completed session */

/*****************************************************************************
 *  calculate vote with input from users
 *****************************************************************************/
void tally()
{
  char  *filename[MAXFILENAME];          /* place to build specific file name */
  int   H[voters+2][n+3];                /* history matrix                    */
  int   i, j, row;                       /* index variables                   */
  array M1;                              /* matrix M1 holds total vote        */
  array M2;                              /* matrix M2 holds average vote      */
  int   O[n+1];                          /* input orderings                   */
  FILE *results;                         /* file for vote analysis            */
  int   split = FALSE;                   /* return flag to caller, zero normally, */
                                         /* 1 if there are 50/50 splits in voting */
  array V;                               /* matrix V holds individual vote    */

  get_history(H);
  strcpy(filename, subject);          /* create vote analysis results file */
  strcat(filename,".results");
  if ( (results = fopen(filename,"w")) == NULL )
  {  perror ("server:  *** results file could not be opened. \n");
     exit(1);
  }
  system("clear");
  fprintf(results,"              SERVER OUTPUT\n");
  fprintf(results,"             of Vote Analysis\n\n");

  blankarray(M1);                          /* begin with zeroed arrays  */
  blankarray(M2);
                                  /* analyze vote history file */
  for (row = 1; row < voters + 1; row ++)
  {
    blankvote(O,V);
    /* convert individual vote to array that lists alternatives in ordering */
    for ( j = 3; j < n+3; j++ )
      O[j-2] = H[row][j];                  /* set up an intermediate array */
                                  /* construct individual vote array */
    for (i=1; i < n+1; i++ )               /* fill arrays with ordering */
      for ( j = i+1; j < n+1; j++ )
        V[O[i]][O[j]] = 1;

    if (DEBUG)                             /* debug printout of array */
    {  printf("Individual vote #%d array: (as used in analysis)\n\n",row);
```

```
      writearray(stdout,V,0,0);
    }
    for (i=1; i < n+1; i++ )              /* add in individual ordering */
      for ( j = 1; j < n+1; j++)          /* to make composite vote array */
        M1[i][j] = M1[i][j] + V[i][j];
  }

    for (i = 1; i < n+1; i++ )            /* construct average vote array */
      for ( j = 1; j < n+1; j++)
        M2[i][j] = M1[i][j]/voters;

    for (i = 1; i < n; i++)               /* check for any 50/50 splits */
      for (j = i+1; j < n+1; j++)
        if (M2[i][j] == M2[j][i])
          split = TRUE;

    fprintf(results,"In the vote matrix, each position shows the ");
    fprintf(results,"numbers of voters who \nprefer the row alternative ");
    fprintf(results,"over the column alternative.  The total\n");
    fprintf(results,"vote looks like this:\n\n");

    writearray(results,M1,0,0);
    fprintf(results,"\n\n  average vote:  \n\n");
    writearray(results,M2,1,0);

    if (split)
    {
     fprintf(results,"A 50/50 split has occurred and a group preference \n");
     fprintf(results,"ordering cannot be constructed from the individual\n");
     fprintf(results," preferences.   \n\n");
     if (DEBUG)
     {
      printf("A 50/50 split has occurred and a group preference ordering\n");
      printf("  cannot be constructed from the individual preferences.\n\n");
     }
     fclose(results);
     H[voters+1][0] = 1;    /* shows group preference has been calculated */
     write_history(H);

    }
    else                          /* apply Hagmann's algorithm to data */
      algorithm(M2,results);          /*  putting analysis in results file */
}  /* end of construction of the total tally */

/*****************************************************************************=
 *  incoming message is from a potential voter
 *****************************************************************************/
void  V_who(buf,finished,H,row,voted,msgsock,day)

  buffer buf;
```

```
  int    finished;                    /* flag for indicating end of voting */
  int    H[voters+2][n+3];            /* history matrix       */
  int    row;                         /* user's history row   */
  int   *voted;                       /* numher who have voted */
  int    msgsock;                     /* socket identifier    */
  double day;                         /* current system day   */
{
  int i, j;                           /* index variables      */
  char thisvote[n];                   /* user's vote          */
  int what;                           /* user's option choice */

  what = huf[1] - '0';                /* extract choice from msg.  */
  if (what == ENTER_VOTE)             /* extract vote from message */
  {
   j = 0;
   for (i=2;i<n+2;i++)
      thisvote[j++] = huf[i];
  }
  serve_user(what, voted, msgsock, H, row, thisvote, day);
    if (*voted == voters)
      finished = TRUE;

/* do if all votes are in and group preference has not been calculated */
    if (finished && (H[voters+1][0] == 0))
    {
      tally();
    }
}     /* end V_who */

/*****************************************************************************
 *  communicate with users who call - handle incoming messages
 *****************************************************************************/
void communicate(msgsock,voter_tahle)
  int msgsock;                        /* socket identifier    */
  struct verify *voter_tahle;
{
  huffer huf;                         /* message buffer           */
  douhle day;                         /* current system day       */
  char   filename[MAXFILENAME];       /* structure to hold filename */
  int    finished;                    /* is voting complete       */
  int    H[voters+2][n+3];            /* history matrix:
                                      * row & col 0 are used only to label array
                                      * H[row][1] holds voter's identification #
                                      * H[row][2] flags is a vote is stored
                                      * H[row][3-(n+3)] stores any vote
                                      */
  int    i;                           /* index variahles          */
  int    key = 0;                     /* users identification     */
```

```
char    loguame[LOGINlen];              /* holds userlogin name       */
int     row;                            /* row # if user matches authorized voter */
int     rval;                           /* function return value      */
struct timeval  tp;                     /* system's GMT time & date   */
struct timezone tzp;                    /* local time zone adjustment */
struct tm       *now;                   /* struct. to hold local time */
int     voted;                          /* number of votes in history */
int     what;                           /* user's choice              */

/* start a communication session */
 bzero(buf, sizeof(buf));
 if ((rval = read(msgsock, buf, 1024)) < 0)
 {  perror("server:  reading stream message");
    exit(1);
 }
 else if ((rval == 0) && (DEBUG))
       printf("Ending connection\n");
 else
 {                                      /* calculate today's date */
    gettimeofday(&tp, &tzp);
    now = localtime(&tp.tv_sec);
    day = 10000*(now->tm_year)+100*(now->tm_mon+1)+(now->tm_mday);

    if (DEBUG)                          /* optional debug printouts */
    {
      printf("day = %.0f\n",day);
      if ( day > deadline)
        printf("Voting has been completed.\n");
      else
        printf("keep on voting.\n");
    }

    voted = get_history(H);             /* bring in vote history */
/* if everyone has voted or the deadline has been reached, voting is finished */
    if ((voted == voters) || (day > deadline))
       finished = TRUE;
    else
       finished = FALSE;
    if (buf[0] == 'V')          /* incoming message is from a vote process */
    {
      what = buf[1] - '0';              /* extract user's option choice */
      if (what == ENTER_VOTE)
        i = n+2;                        /* enter_vote has a long message */
      else                              /* all other messages are shorter */
        i = 2;                          /* extract user id. from message  */
      if ((rval = sscanf(&buf[i],"%4d%s",&key,logname)) < 0)
      {   perror("server:  sccanf");
          exit(1);
      }
```

```
if(DEBUG)  printf("server-->voter %s, key = %d\n", logname, key);
row = 0;
for (i=0; i<voters+1; i++)    /* see if user is in authorized list */
{
  if ((rval = strcmp(voter_table[i].user,logname)) == 0)
     if ((rval = strcmp(voter_table[i].votekey,key)) == 0)
     {  if (i==0)
           row = 99;       /* creator's id. is in row 0, flag as 99 */
        else
           row = i;
        break;
     }
}
if (!row)
{                    /* unauthorized user - return no information */
   sprintf(buf," 0 0 0                ");
   if (write(msgsock, buf, strlen(buf)) < 0)
   {   perror("server:  writing on stream socket");
       exit(1);
   }
}
if  (row == 99)
{              /* creator - ask if voting object should be retired */
   sprintf(buf,"%d 0 0                ",row);
   if (write(msgsock, buf, strlen(buf)) < 0)
   {  perror("server:  writing on stream socket");
      exit(1);
   }
   if ((rval = read(msgsock, buf, 1024)) < 0)
   {  perror("server:  reading stream message");
      exit(1);
   }
   if (buf[0]=='Y')
   {  strcpy(filename,subject);
      if ((rval = unlink(filename))< 0)        /* remove server */
      {  perror("server:  in unlinking voting object");
         exit(1);
      }
                 /* save support files in log:  */
      /* copy & remove  .autbor, .history, .log & .results files */
      sprintf(buf,"cat %s.* >> temp.%s.log",subject, subject);
      system(buf);
      sprintf(buf,"/bin/rm %s.*",subject);
      system(buf);
      /* copy and remove subject_alt.menu & subject_back.A...   */
      sprintf(buf,"cat %s_* >> temp.%s.log",subject, subject);
      system(buf);
      sprintf(buf,"/bin/rm %s_*",subject);
      system(buf);
```

```
            /* move logged files to expired sub-directory */
            sprintf(buf,"mv temp.%s.log expired/%s.log.%.0f",
                        subject,      subject, day);
            system(buf);
            sprintf(buf,"'%s%s\n %s\n '%s.log.YYMMDD' %s\n", subject,
             "' is now obsolete.  Information regarding this voting object",
             " is stored with other expired objects under the filename",
              subject, "where YYMMDD is today's date.\n\n");

            if (write(msgsock, buf, strlen(buf)) < 0)
            {  perror("server:  writing on stream socket");
              exit(1);
            }
            exit(0);
          }
          else                  /* creator does not want to retire server */
            strcpy(buf,"Goodbye - no change in status");
          if (write(msgsock, buf, strlen(buf)) < 0)
          {  perror("server:  writing on stream socket");
            exit(1);
          }
        }
        V_who(buf,finished,H,row,&voted,msgsock,day);
    }
    else
      if (buf[0] == 'F')   /* fileserver message regarding a user's comment */
      {
        if (DEBUG)  printf("FS has a file to concatenate\n");
        sprintf(filename,"cat %s >> %s_back.%c", &buf[2], subject, buf[1]);
        if (DEBUG)  printf("filename = %s", filename);
        system(filename);
        sprintf(filename,"/bin/rm %s", &buf[2]);
        system(filename);
      }
    }         /* end of else- successful read */
    close(msgsock);
}  /* end of communication */

/*****************************************************************************/
/*****************************************************************************/
main()
{

  buffer buf;                           /* temporary storage buffer    */
  FILE  *fd;                            /* file descriptor             */
  char  *filename[MAXFILENAME];         /* structure to hold filenames */
  int    i,j;                           /* index variables             */
  int    msgsock;                       /* socket identifier           */
  fd_set ready;                         /* socket check variable       */
  int    rval;                          /* function return value       */
```

```
int    sock;                                  /* socket identifier          */
struct timeval  to;                           /* system clock structure     */
struct verify  *voter_table;

      /*  sets up the character conversion table for vote recognition. */
strcpy(&buf[1],"ABCDEFGHIIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");
for (i=1,j=27; i<27; i++,j++)
{ convert[buf[i]] = i;
  convert[buf[j]] = i;
};


            /* set up table of authorized voters and associated key */
strcpy(filename, subject);
strcat(filename,".author");
if ((fd = fopen(filename,"r")) < 0)
{  perror("server:  opening authorization file");
   exit(1);
}
voter_table = (struct verify *)calloc(voters+1, sizeof(struct verify));

for (i=0;i<voters+1;i++)
{
   if ((rval = fscanf(fd,"%d %s %*s",
              &voter_table[i].votekey, voter_table[i].user)) < 0)
   {  perror("server:  getting votekey & user");
      exit(1);
   }
}
fclose(fd);

sock = socket_setup();                  /* establish a socket connection */
do {
     FD_ZERO(&ready);
     FD_SET(sock, &ready);

/*  This program uses select() to check that someone is trying to connect
 *  before calling accept().  Select is synchronous i/o multiplexing.
 */
     if (select(sock + 1, &ready, 0, 0, (char *)0) < 0)
     {  perror("server:  select");
        continue;
     }
     if (FD_ISSET(sock, &ready))            /* ready to accept a caller */
     {  msgsock = accept(sock, (struct sockaddr *)0, (int *)0);
        if (msgsock == -1)
        {  perror("server:  accept");
           exit(1);
        }
        else
```

```
            communicate(msgsock, voter_table);    /* handle communication */
        }
    } while (TRUE);        /*  after vote is completed it continues to give
                            *  out results until the adminstrator removes
                            *  process from system */
}   /* end of main */

/*****************************************************************************/
/******** Nancy J. Calhoun ***************************************************/
/*********************************************** September, 1988 **********/
/*****************************************************************************/
```

APPENDIX G

PREFER.fileserver.c Source Program

```
/*****************************************************************************
    This program was written as part of the implementation of a Small Group Deci-
    sion Support System designed by C. Hagmann. It implements a distributed
    voting server called PREFER. The programming was done by Nancy Calhoun
    as part of a Master's Report project in the summer of 1988.

    This is the source file for the PREFER.fileserver module. It is a generic file
    server which acts as a transmitter of resource files to designated voters. The
    voting server creates a specific file server whenever a voter requests to append
    comments to background files or to receive information held in files on the
    server's computer.
 *****************************************************************************/

 /* paths to include files vary so check the requirements of your system */
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <sgtty.h>

/*---------------------- PREFER definitions ------------------------*/
#define  DEBUG          1          /* if debugging, set to 1 for message output */
#define  FALSE          0
#define  TRUE           1
#define  MAXFILENAME    256        /* maximum length for filenames */
#define  BACKGROUND     2          /* user options handled by fs.  */
#define  GET_RESULTS    3
#define  COMMENT        5

/*---------------------- end of definitions ------------------------*/


/*****************************************************************************
 *  sets up the socket on this end of the interprocess communication.
 *****************************************************************************/
int  sckt_setup(host,port)
  char *host;                       /* host computer network name */
  int   port;                      /* socket id.                 */
{
 struct hostent *hp, *gethostbyname();
 struct sockaddr_in server;
```

```
 int sock;                              /* communications socket id */

 sock = socket(AF_INET, SOCK_STREAM, 0);              /* Create socket */
 if (sock < 0)
 { perror("fs:  opening stream socket");
   exit(3);
 }
 server.sin_family = AF_INET;          /* AF_INET = ARPA internet addresses */
 hp = gethostbyname(host);      /* get network id of host name sent by user */
 if (hp == 0)
 {  fprintf(stderr, "%s: unknown host", host);
   exit(3);
 }
 bcopy(hp->h_addr, &server.sin_addr, hp->h_length);
 server.sin_port = htons(port);                  /* htons = host byte order to network */
                                                 /*    byte order short integer        */
                                                 /* connect to specified socket:       */
 if (connect(sock, &server, sizeof(server)) < 0)
 {  perror("fs:  connecting stream socket setup");
   exit(3);
 }
 return(sock);                          /* return socket identifier  */
}   /* end of sckt_setup */

/*****************************************************************************
 *  receive the file over the socket
 *****************************************************************************/
char  receive_file(sock,filename)

 int  sock;                                      /* socket id.      */
 char *filename[MAXFILENAME];                    /* structure for building filenames */
{
 char   buf[1024];                               /* message buffer  */
 FILE   *fd;                                     /* file descriptor */
 int    listflag = TRUE;                         /* used to signal end of file transfer */

 fd = fopen(filename,"w");
 if (DEBUG)                                          /* diagnostic print */
   printf("%s has just been opened for w\n",filename);
 while (listflag)                       /* catch detail file and store locally */
 {
   bzero(buf,1024);                             /* clean slate for next communication */
   if (read(sock, buf, 1024) < 0)
   {   perror("fs:  reading message");
       exit(3);
   }
   fprintf(fd,"%s",&buf[2]);                    /* store what server sent */
   if (buf[0] == '0')
     listflag = FALSE;                             /* have completed list    */
```

```
  }
  fclose(fd);
  return(buf[1]);              /* alternative id was stored in 2nd position */
}   /* end of receive_file */

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

main(argc,argv)
  int  argc;
  char *argv[];
{
  char      buf[1024];                  /* message buffer        */
  char      *filename[MAXFILENAME];     /* structure for building filenames */
  FILE      *fd;                        /* file descriptor       */
  FILE      *fs;                        /* file descriptor       */
  int       listflag;                   /* used to signal end of file transfer */
  char      bost[10];                   /* computer network name */
  int       msgsock;                    /* socket identifier     */
  int       port;                       /* socket identifier     */
  int       rval;                       /* function return value */
  fd_set    ready;                      /* socket information    */
  int       sock, sock2;                /* socket identifiers    */
  int       length;                     /* socket information    */
  struct    sockaddr_in server;         /* socket information    */
  struct    timeval to;                 /* system clock value    */
  int       what;                       /* what the caller wants */
  int       which;                      /* to which alternative this comment belongs */

/* this process is tbe child of the voting server daemon.  Since it is much
 * shorter lived than it's parent, if it does its work directly, it becomes
 * a zombie after dying.  To prevent zombies from clogging up things this
 * process immediately forks off a child to do the work and it dies.  The
 * "grandchild" will not become a zombie when it finishes tbe work because
 * it will be taken care of by tbe system.
 */

 if (fork() !=0);                        /* parent process - does not wait */
    /* empty option */
 else                    /* child process - does all the work of fileserver */
 {
   /* Create socket */
   sock = socket(AF_INET, SOCK_STREAM, 0);
                  /* AF_INET is ARPA internet addresses.  SOCK-STREAM  */
                  /*   is sequenced 2-way connection based byte stream */
   if (sock < 0)
   {  perror("fs:  opening stream socket");
      exit(3);
   }
```

```
    server.sin_family = AF_INET;                /* Name socket using wildcards */
    server.sin_addr.s_addr = INADDR_ANY;        /* accept call from anyone     */
    server.sin_port = 0;                        /* let system assign socket id */
    length = sizeof(server);
    if (bind(sock, &server, length))            /* bind a name to the socket   */
    {  perror("fs:  binding stream socket");
       exit(3);
    }
                            /* Find out assigned port number and print it out */
    if (getsockname(sock, &server, &length))
    {  perror("fs:  getting socket name");
       exit(3);
    }
    if ( (fs = fopen("prefer.fs","w")) == NULL )
    {  perror ("fs:  socket id file could not be opened. \n");
       exit(3);
    }
                    /* put socket info. in file for voting user to read */
    fprintf(fs,"%d",ntohs(server.sin_port));
    fclose(fs);

                                                /* Start accepting connections */
    listen(sock, 1);                            /* will make one contact */
    ioctl(0,TIOCNOTTY,0);                       /* system call so daemon will not */
                                                /*  continue to tie up a terminal */
    if (DEBUG)
       printf("fs: Socket has port #%d\n", ntohs(server.sin_port));
    else
    {  fclose(stdin);
       fclose(stdout);
       fclose(stderr);
    }
/*
 * This program uses select() to check that someone is trying to connect
 * before calling accept().
 */
    FD_ZERO(&ready);
    FD_SET(sock, &ready);
    to.tv_sec = 5;                      /* fileserver will check regularily */
    if (select(sock + 1, &ready, 0, 0, &to) < 0)
    {  perror("fs:  select");
       exit(3);
    }
    if (FD_ISSET(sock, &ready))
    {
       msgsock = accept(sock, 0, 0);
       if (msgsock == -1)
       {  perror("fs:  accept");
          exit(3);
```

```
}
else
  do {
      bzero(buf, sizeof(buf));
      if ((rval = read(msgsock, buf, 1024)) < 0)
      {  perror("fs:  reading 2nd stream message");
         exit(3);
      }
      else if (rval == 0)
      {  if (DEBUG)  printf("\nEnding 2nd connection\n"); }
      else
      {
        what = buf[0] - '0';        /* extract caller's choice      */
        if (what == COMMENT)        /* caller wants to send comment */
        {
          sscanf(buf,"%d %d %s %s ",&what,&port,host,filename);
          which = receive_file(msgsock,filename);    /* get comment */
          sprintf(buf,"%c%c%s",'F', which, filename);

              /* let vote server append the comment to the proper */
              /*   file so establish socket to vote server       */
          sock2 = sckt_setup(host,port);
          if (write(sock2, buf, 1024) < 0)
          {  perror("fs: writing on stream socket to server");
             exit(3);
          }
          close(sock2);
        }
        else
        {  if (what == BACKGROUND)
           {              /* caller wants to get a background file */
             strcpy(filename,argv[1]);          /* build file name */
             strcat(filename,"_back.");
             strcat(filename,&buf[1]);          /* which alternative */
           }
           else
             if (what == GET_RESULTS)
             {            /* caller wants to get vote analysis file */
               strcpy(filename,argv[1]);
               strcat(filename,".results");
             }

           fd = fopen(filename,"r");
           listflag = TRUE;
           while (listflag)                          /* read list */
           {
             bzero(buf, sizeof(buf));
             if ( fd == NULL )
             { strcpy(buf,"0** resource file could not be opened.\n");
```

```
                    listflag = FALSE;
                  }
                  else
                    if ((listflag=fread(&buf[1],sizeof(char),1022,fd)) <0)
                    {   perror("fs: reading background file");
                        exit(3);
                    }
                    else
                    {
                     if (listflag < 1022)
                       /* tell user whether another read is unnecessary */
                     {  listflag = FALSE;
                        buf[0] = '0';
                     }
                     else
                     { buf[0] = '1';
                       buf[1023] = '\0';
                     }
                    }
                  if (write(msgsock, buf, 1024) < 0)
                  { perror("fs:  sending background over stream socket");
                    exit(3);
                  }
                  bzero(buf, sizeof(buf));
                }
                fclose(fd);
              }
            }
          } while (rval != 0 );
          close(msgsock);
    }   /* if socket is ready */
  }   /* end of child process */
}   /* end of file server */

/*****************************************************************************/
/******** Nancy J. Calhoun, programmer ***********************************/
/*************************************** September, 1988 ************/
/*****************************************************************************/
```

APPENDIX H

PREFER.vote.c Source Program

```
/*****************************************************************************
    This program was written as part of the implementation of a Small Group Deci-
    sion Support System.  It is the implementation of part of the SGDSS design by
    C. Hagmann.  The programming was done by Nancy Calhoun as part of a
    Master's Report project in the summer of 1988.

    This PREFER.vote module is a generic process to act as the voter's interface to
    the vote server.  Arguments given on the command line must supply the infor-
    mation needed to connect to the proper server process.

    This program creates a socket and initiates a connection with the socket number
    given in the command line.  The form of the command line is:
                PREFER  topic  hostname  portnumber  userIDnumber
 *****************************************************************************/

/*---------------------- include files required for definitions --------*/
/* paths to include files vary so check the requirements of your system  */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <ctype.h>
#include <strings.h>                /* needed for index function pointer */
#include <sys/wait.h>

#define  MORE       "/usr/uch/more"
#define  VI         "/usr/uch/vi"

/*----------------------------- basic definitions ---------------------*/
#define  FALSE              0
#define  TRUE               1
#define  MAXalt             9          /* maximum # of alternatives allowed */
#define  MAXFILENAME        256        /* maximum size of filenames         */

/*----------------------------- message buffer choice flags ----------*/
#define  LEAVE              0
#define  ENTER_VOTE         1
#define  BACKGROUND         2
#define  GET_RESULTS        3
#define  CHECKIN            4
#define  COMMENT            5
```

```
#define  ERASE              6

/*------------------------------ size limitations ---------------------*/
#define  BUFFERSIZE     1024                          /* size of message buffer  */
#define  IDlen          4                             /* digits in id. number    */
#define  LOGINlen       8+1                           /* length of login name    */
#define  L_MSGlen       2+MAXalt+IDlen+LOGINlen       /* length of long message  */
#define  S_MSGlen       2+IDlen+LOGINlen              /* length of short message */

/*------------------------------ command line constants ---------------*/
#define  ARGcount       5          /* # arguments on line    */
#define  TOPICpos       1          /* topic word position    */
#define  HOSTpos        2          /* host computer name pos.*/
#define  SOCKpos        3          /* socket number position */
#define  IDpos          4          /* user's id. # postion   */

/*------------------------------ type defination and globals --------*/
typedef char            buffer[BUFFERSIZE];
typedef struct          vote_rec
                          {  int  n;              /* number of alternatives    */
                             int  finished;       /* if all votes are in       */
                             int  havevote;       /* if this user has voted    */
                             char *prev_vote;     /* user's previous vote      */
                             char *group_vote;    /* the group preference vote */
                          };
int  convert[256];                    /* vote letter/number conversion tables */
static cbar  ntc[MAXalt+1] = {' ','A','B','C','D','E','F','G','H','I'};
                                      /* ntc - number to letter conversion */

/*************************************************************************
 *  sets up the character conversion table for vote recognition.
 *************************************************************************/
void  table_setup()
{
 buffer  buf;                                 /* storage buffer     */
 int     i, j;                                /* indexing variables */


 for (i=0; i<256; i++)
   convert[i] = 0;

 strcpy(&buf[1],"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");
 for (i=1,j=27; i<27; i++,j++)
 { convert[buf[i]] = i;
   convert[buf[j]] = i;
 };
```

```
}  /* end of table_setup */

/*************************************************************************
 *  sets up the socket on this end of the interprocess communication.
 *************************************************************************/
int  sckt_setup(host, port)

 char      *host;                        /* pointer to host name */
 int       port;                         /* socket identifier    */
{
 struct    hostent *hp;                  /* socket functions     */
 struct    hostent *gethostbyname();
 struct    sockaddr_in server;
 int       sock;                         /* communications socket */


 sock = socket(AF_INET, SOCK_STREAM, 0);        /* Create socket */
                          /* AF_INET = ARPA internet addresses.  */
        /* SOCK_STREAM = sequenced 2-way connection based byte stream */
 if (sock < 0)
 { perror("v:  opening stream socket");
   exit(2);
 }
 server.sin_family = AF_INET;                /* Connect socket using name */
 hp = gethostbyname(host);                   /* specified by command line. */
 if (hp == 0)
 {  fprintf(stderr, "%s: unknown host", host);
   exit(2);
 }
 bcopy(hp->h_addr, &server.sin_addr, hp->h_length);
 server.sin_port = htons(port);
                                        /* htons = host byte order to network */
                                        /*       byte order short integer     */
                                        /* atoi = covert string to integer    */
 if (connect(sock, &server, sizeof(server)) < 0)
 {  perror("v:  connecting stream socket");
   exit(2);
 }
 return(sock);
}   /* end of sckt_setup */

/*************************************************************************
 *  used to get any one letter response from user
 *************************************************************************/
int get_choice()
{ char  opt;                            /* user's first input character */
  char  eat;                            /* eats rest of input line      */

 opt = getchar();
 while ((opt == '\n') || isspace(opt))
```

```
    opt = getchar();
  eat = getchar();
  while (eat != '\n')
    eat = getchar();
  return(opt);
}   /* end of get_choice */

/*************************************************************************
 *  receive the file over the socket
 *************************************************************************/
void  receive_file(sock2)
  int  sock2;
{
  int     listflag = TRUE;
  FILE   *fd;
  buffer  buf;
  union wait status;                                  /* child status */

  if ( (fd = fopen("tmp.vote.detail","w")) == NULL )
  {  perror ("v:  socket id file could not be opened. \n");
     exit(2);
  }
  while (listflag)                    /* catch detail file and store locally */
  {
     bzero(buf,1024);               /* clean slate for next communication */
     if (read(sock2, buf, 1024) < 0)
     {   perror("v:  reading message");
         exit(2);
     }
     fprintf(fd,"%s",&buf[1]);                      /* store what server sent */
     if (buf[0] == '0')
       listflag = FALSE;                           /* have completed list    */
  }
  fclose(fd);                    /* use "more" to read the detail file that is */
                                 /*     temporarily stored in user's directory */
  if (fork() !=0)                                  /* parent process */
  {   if ((wait(&status)) == -1)
     {  perror("v:  wait for child");
        exit(2);
     }
  }
  else    /* child process */
     if ((execl(MORE, MORE, "-20", "tmp.vote.detail", 0))==-1)
     {  perror("v:  cannot start second process \n");
        exit(2);
     }
  if ((unlink("tmp.vote.detail"))< 0)              /* destroy temporary file */
  {  perror("v:  in unlinking tmp.vote.detail");
     exit(2);
```

```
  }
    printf("\n------- end of resource file  ****  Press <RETURN> key to continue. -----");
    getchar();
    system("clear");

}   /* end of receive_file */

/*************************************************************************
 *  create socket for receiving file details from file server
 *************************************************************************/
void read_details(which,n,host,sockptr)

  int       wbich;                  /* where output should be printed */
  int       n;                      /* number of alternatives         */
  char      *host;                  /* socket information             */
  int       *sockptr;               /* pointer to current socket #    */
{
  buffer    buf;                    /* message buffer                 */
  int       flag;                   /* flag for                       */
  int       opt;                    /* user's option                  */
  int       port;                   /* socket port identification     */
  iut       sock2;                  /* another communications socket  */
  int       temp;                   /* flag for                       */

  bzero(buf,sizeof(buf));           /* clean slate for next communication */
  if (read(*sockptr, buf, 1024) < 0)
  {   perror("v:  reading message");
      exit(2);
  }
  close(*sockptr);

  port = atoi(buf);
  sock2 = sckt_setup(host,port);        /* set up socket to talk to fileserver */

  if (which == BACKGROUND)              /* user wants to read a background file */
  {  printf("\nWhat alternative do you want described?  (A - %c)  ",
                                                    ntc[n]);
     do
     {  opt = get_choice();
        temp = convert[opt];                    /* get number of choice    */
        if ((temp < 1) || (temp > n))           /* is it in correct range? */
        {  printf("Enter your choice again.  \007");
           temp = 0;
        }
     } while (!temp);
     printf("\n\n");
     bzero(buf,sizeof(buf));            /* clean slate for next communication */
     sprintf(buf,"2%c",ntc[temp]);              /* opt in capital form  */
     if (write(sock2, buf, sizeof(buf)) < 0)
     {  perror("v:  writing on stream socket");
```

```
      exit(2);
    }

    printf("\n\nPlease wait for resource file to be sent.\n\n");
    receive_file(sock2);
  }
  else if (which == GET_RESULTS)          /* user wants to see results file */
  {
    bzero(buf,1024);                      /* clean slate for next communication */
    buf[0] = '3';
    buf[1] = '\0';
    if (write(sock2, buf, sizeof(buf)) < 0)
    { perror("v:  writing on stream socket");
      exit(2);
    }
    printf("\n\n\nPlease wait for analysis file to be sent.\n\n\n");
    receive_file(sock2);
  }
  close(sock2);
}    /* end of read_details */

/*************************************************************************
 *  identifies user on this end to server on the other end of connection.
 *************************************************************************/
void  identify(s_msg,v,sockptr)
 char  s_msg[S_MSGlen];
 struct vote_rec *v;
 int  *sockptr;
{
 buffer    buf;                        /* message buffer          */
 int       dn;                         /* notation for  (2*v->n)  */
 int       flag;                       /* set when user enters incorrect response */
 char      opt;                        /* users option            */

 if (write(*sockptr, s_msg, strlen(s_msg)) < 0)
 { perror("v:  writing on stream socket");
   exit(2);
 }
 bzero(buf,sizeof(buf));                          /* clean the slate */
 if (read(*sockptr, buf, 1024) < 0)
 {  perror("v:  reading message");
    exit(2);
 }                                    /* does server recognize this user? */
 if ((sscanf(buf,"%d %d %d", &v->n, &v->finished, &v->havevote)) <= 0)
 { perror("v:  No conversion of string input");
   exit(2);
 }
 if (v->n == 0)
```

```
{  printf("********* You are not an authorized user. **********\n");
   close(*sockptr);
   exit(1);
}
else if (v->n == 99)
{  printf("Since you are the creator, do you want to remove the\n");
   printf("  voting server at this time?  (Y/N)  ");
   do
   {
     opt = get_choice();
     flag = FALSE;
     if ((index("YyNn",opt)) == NULL)
     {  printf("   Please choose 'Y' or 'N'.  \007");
        flag = TRUE;
     }
     else
     {  if ((opt == 'y') || (opt == 'Y'))
           strcpy(buf,"Y ");
        else
           strcpy(buf,"N ");
        if (write(*sockptr, buf, strlen(buf)) < 0)              /*  send it */
        {  perror("v:  writing on stream socket");
           exit(2);
        }
        if (read(*sockptr, buf, 1024) < 0)
        {   perror("v:  reading message");
            exit(2);
        }
        printf("%s",buf);                                  /* print server's reply */
        exit(0);
     }
   }  while (flag);
}

else                    /* authorized user gets more information from server */
{  dn = 2*v->n;                         /* dn used to make notation easier */
                                          /* get memory space for votes      */
   if ((v->prev_vote = (char *) calloc(dn+2,sizeof(char))) == NULL)
   {  perror("v:  allocating memory for string");
      exit(2);
   }
   if ((v->group_vote = (char *) calloc(2*MAXalt,sizeof(char))) == NULL)
   {  perror("v:  allocating memory for string");
      exit(2);
   }
   strncpy(v->prev_vote,&buf[6],dn);
   strncpy(v->group_vote,&buf[6+dn+1],2*MAXalt);
}
} /* end of identify */
```

```
/*************************************************************************
 *  server displays the alternative listing to user
 *************************************************************************/
int  alt_menu(v,sockptr)
 struct vote_rec  *v;
 int  *sockptr;
{
 buffer    buf;                        /* message buffer              */
 int       flag;                       /* flags incorrect user input  */
 int       listflag = TRUE;            /* signals end of file transfer */
 int       opt;                        /* user's option               */

 wbile (listflag)
 {
   printf("\n\n");
   bzero(buf,1024);                          /* clean the slate */
   if (read(*sockptr, buf, 1024) < 0)
   {   perror("v:  reading message");
       exit(2);
   }
   printf("%s",&buf[1]);                      /* print list server sent */
   if (buf[0] == '0')  listflag = FALSE;      /* have completed list    */
 }
 close(*sockptr);

 bzero(buf,1024);                          /* clean the slate */
 if (v->finished)
 {
    printf("\n\n");
    printf(" The voting has been completed:  ");
    printf("group preference order - %s\n", v->group_vote);
    printf("%45c your vote - %s\n",' ', v->prev_vote);
 }
 while (!v->finished)
 {
  printf("\nChoose:  D(descriptions), V(vote), P(pro/con comments), \n");
  printf("              or Q(quit to Op.Sys.) - (_)\b\b");
  opt = get_choice();
  if ((index("Qq",opt)) != NULL)
     return(LEAVE);

  if ((index("Dd",opt)) != NULL)
     return(BACKGROUND);

  if ((index("Pp",opt)) != NULL)
     return(COMMENT);
```

```
    if ((index("Vv",opt)) != NULL)
          return(1);
    else printf("\r\007");                    /* unacceptable character */
  }       /* end while !finished */
  if (v->finished)
  {
    printf("\nDo you want to see the group preference analysis?  (Y/N) ");
    do
    {
      opt = get_choice();                     /* capture user response */
      flag = FALSE;
      if ((index("YyNn",opt)) == NULL)
      {  printf("   Please choose 'Y' or 'N'.  \007");
        flag = TRUE;
      }
      else if ((opt == 'y') || (opt == 'Y'))
        return(GET_RESULTS);
      else
        return(LEAVE);
    } while (flag);
  }                 /* end of if finished */
  return(LEAVE);
}                 /* end of alt_menu */

/*************************************************************************
 *  check to see if user wants to change the recorded vote
 *************************************************************************/
int no_change(prev_vote)
  char *prev_vote;
{  int opt;

  printf("\nYou have already voted.  ");
  printf("Your previous vote:  %s\n",prev_vote);
  printf(" Do you want to change your vote.\007 (Y/N)  ");
  do
  {  opt = get_choice();
    if ((index("YyNn",opt)) == NULL)
      printf("    Please choose 'Y' or 'N'.  \007");
    else if ((opt == 'y') || (opt == 'Y'))
      return(FALSE);
    else
      return(TRUE);
  } while (TRUE);
}                   /* end of no_change */

/*************************************************************************
 * check this vote for duplicate & out-of-range preferences
 *************************************************************************/
int incorrect(n,l_msg)
```

```
 int        n;
 char       l_msg[L_MSGlen];          /* contains users vote          */
{
 int        value;                    /* integer representation of vote */
 int        flag = FALSE;             /* flags incorrect user input     */
 int        i, j;

 for (i = 2; i < n+1; i++)
 {  for (j = i+1; j < n+2; j++)
      if (l_msg[i] == l_msg[j])
      {  flag = TRUE;
         printf("   Duplicate '%c' entry, please reenter ordering:\007  ",
                        l_msg[i]);
      }
 }
 for (i = 2; i < n+2; i++)
 {
   value = convert[l_msg[i]];
   if ((value > n) || (value < 1))
   { flag = TRUE;
     printf("   Entry '%c' out of range, please reenter ordering:\007  ",
                    l_msg[i]);
   }
  }
  return(flag);
}                   /* end of incorrect */

/*************************************************************************
 *  get users vote from stdin
 *************************************************************************/
void receive_vote(l_msg,n)
 char  l_msg[L_MSGlen];
 int   n;
{
 int        check;                    /* flags incorrect user input */
 int        count;                    /* count of letters entered   */
 int        flag;                     /* flags incorrect user input */
 int        i, j;                     /* index variables            */
 buffer     opinion;                  /* catch user's vote          */
 int        opt;                      /* user option                */

 flag = FALSE;
 l_msg[1] = ENTER_VOTE + '0';         /* indicates a vote is included */
                              /*     building a vote message  */
 printf("Type in the alternative letters in the");
 printf(" order of your preference.\n\n");
 printf(" Please use all letters A through %c:  ",ntc[n]);
 do
```

```
{  count = 0;
   gets(&opinion[1]);

   for (i=1, j=2; i< 1024; i++,j++)                    /* prepare to send vote */
   {  while (isspace(opinion[i]))   i++;
      if (opinion[i] != '\0')              /* have not encounted end of input */
      {
        if (count < n)
        {  l_msg[j] = opinion[i];                    /* compact vote to send */
           count++;
        }
        else  break;                              /* vote has become too long */
      }
      else  break;                                      /* end of input  */
   }
   if (( count != n) || (opinion[i] != '\0'))                /* invalid input */
   {  printf("   Your vote has the wrong number of entries, ");
      printf("please try again. \007 ");
      flag = TRUE;
   }
   else
      flag = incorrect(n,l_msg);          /* incorrect returns 1-bad, 0-good */

   if (!flag)                                  /* no obvious errors in input */
   {
      printf("\n%25c Is '",' ');                  /* echo vote entered to user  */
      for (i=2; i<n+2; i++)
         printf("%c ",l_msg[i]);
      printf("' correct? (Y/N)  ");
      do
      {  opt = get_choice();
         check = FALSE;
         if ((index("YyNn",opt)) == NULL)
         {  printf("    Please choose 'Y' or 'N'.  \007");
            check = TRUE;
         }
         else if ((opt == 'n') || (opt == 'N'))
         {  flag = TRUE;                          /* allows reentry of vote */
            printf(" Your preference order is:  ");
         }
         else
            printf("\n\n            THANK YOU FOR VOTING.\n");
      } while (check);
   }            /* end if (!flag) */
 } while (flag);
}               /* end of receive_vote */

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

```
 *  user inputs desired ordering of alternatives
 ************************************************************************/
int  get_vote(l_msg,v)
   char  l_msg[L_MSGlen];                    /* long message       */
   struct vote_rec  *v;
{
   if (v->havevote)                        /*   has user already voted? */
   {
     if (no_change(v->prev_vote))
        return(0);                          /* nothing to be done */
     else
        receive_vote(l_msg,v->n);           /* user wants to change vote */
   }
   else
        receive_vote(l_msg,v->n);                 /* get new vote       */
   return(1);
}    /* end of get_vote */

/************************************************************************/
/*  send the comment over the socket to a file server                   */
/************************************************************************/
void  send_file(sockptr, n, filename)
   int    *sockptr;
   int    n;
   char  *filename;
{
   buffer    buf;                      /* message buffer   */
   FILE      *fd;                      /* file descriptor  */
   int       listflag;                 /* signals the end of file transfer */
   int       opt;                      /* user option      */
   int       pid;                      /* child process id */
   int       rval;                     /* function return value */
   union     wait status;              /* child status     */
   int       temp;                     /* number representation of user choice */

   printf("To what alternative do you want to add a comment?  (A - %c) ",
                                             ntc[n]);
   do
   {  opt = get_choice();
      temp = convert[opt];                    /* get number of choice    */
      if ((temp < 1) || (temp > n))           /* is it in correct range? */
      { printf("Enter your choice again.  \007");
        temp = 0;
      }
   } while (!temp);

   if ((fd = fopen(filename,"w")) == NULL)
   {  perror("** resource file could not be opened for writing.");
```

```
  exit(2);
}
else
{                       /* put blank line into file it is not empty */
  fprintf(fd,"  \n");
  fclose(fd);
}

printf("\n");
printf("A file will be opened for you to insert the \n");
printf("comment on alternative %c (using 'vi').\n\n", opt),
printf("   Press <RETURN> key to continue.   ");
getchar();
if (fork() !=0)                                 /* parent process */
{
  if ((pid = wait(&status)) == -1)
  {  perror("v:  wait for child");
     exit(2);
  }
}
else    /* child process - create file using "vi" */
  if ((rval=execl(VI,VI,filename,0))==-1)
  {  perror("v:  cannot start second process \n");
     exit(2);
  }
printf("\n\n\nPlease wait for comment file to be sent.\n\n\n");
if ((fd = fopen(filename,"r")) == NULL)
{  perror("** resource file could not be opened for sending.");
   exit(2);
}
listflag = TRUE;
while (listflag)                                /* read list */
{
  bzero(buf, sizeof(buf));
  buf[1] = ntc[temp];                   /* choice in capital letter */
  if ((listflag = fread(&buf[2], sizeof(char), 1021, fd)) < 0)
  {  perror("v:  reading comment file");
     exit(2);
  }
  else
  {
     if (listflag < 1021)       /* tells fs that this is end of file */
     {  listflag = FALSE;
        buf[0] = '0';
     }
     else           /* tell fileserver that another read is necessary */
     {  buf[0] = '1';
        buf[1023] = '\0';
     }
```

```
    }
    if (write(*sockptr, buf, strlen(buf)) < 0)
    {  perror("v:  sending comment file to file server on stream socket");
       exit(2);
    }
    bzero(buf, sizeof(buf));
  }
  fclose(fd);
      if ((unlink(filename))< 0)              /* remove temporary file */
      {  perror("v:  in unlinking k#.comment");
         exit(2);
      }

}   /*  end of send_file */


/*************************************************************************/
main(argc, argv)
  int    argc;
  char  *argv[];
{

  buffer    buf;                    /* message buffer          */
  int       flag;                   /* used to control menu loop */
  int       fs_socket;              /* fileserver socket id    */
  char      host[10];               /* host computer name      */
  int       i;                      /* index variable          */

  char     *keep[ARGcount];
  char     *keep2[ARGcount];
             /* keep & keep2:      * keep[0]                  = "prefer"
                                   * keep[TOPICpos]           = topic
                                   * keep[HOSTpos]            = hostmachine
                                   * keep[SOCKpos]            = socket #
                                   * keep[IDpos]              = user's ID #
          */

  char      me[80];                  /* 80 character line        */
  int       n;                       /* number of alternatives   */
  int       opt;                     /* user's option            */
  char      l_msg[L_MSGlen];         /* two sizes of messages:   */
  char      s_msg[S_MSGlen];         /*       long and short     */
  int       server_socket;           /* server socket ident.     */
  int       sock, sock2;             /* communications sockets   */
  char      topic[80];               /* topic title for menu     */
  iut       user_id;                 /* user identification #    */
  struct    vote_rec *v;             /* voting status information */
  char      filename[MAXFILENAME];
```

```
v = (struct vote_rec *) malloc (sizeof(struct vote_rec));
server_socket = atoi(argv[SOCKpos]);
strcpy(topic,argv[TOPICpos]);
strcpy(host,argv[HOSTpos]);
user_id = atoi(argv[IDpos]);

v->finished = FALSE;                          /* initial settings */
v->havevote = FALSE;

table_setup();
                        /* open first socket to vote server */
sock = sckt_setup(host, server_socket);

printf("\n\nv--> Enter your login name:  ");
if ((scanf("%s",me)) != 1)
{  perror("v:  user input");
   exit(2);
}
sprintf(s_msg,"V%d %d %s",CHECKIN,user_id,me);
identify(s_msg,v,&sock);              /* identify process to vote server */
n = v->n;                              /* n used to simplify notation     */
                /* build long message for vote:     V9ABC..n9999XXXX    */

                                       /*     consisting of     */
l_msg[0] = 'V';                        /*  'V' to flag voter    */
l_msg[1] = ENTER_VOTE + '0';           /*   what - ENTER_VOTE   */
  /*   (entered later) */              /*   alternatives        */
strcpy(&l_msg[n+2],argv[IDpos]);       /*   user ID #           */
strcpy(&l_msg[n+6],me);                /*   login name          */


do
{  flag = FALSE;                                  /* get alternative menu  */
   if ((opt = alt_menu(v,&sock)) == ENTER_VOTE)
   {  opt = get_vote(l_msg,v);                   /* user wants to vote */
      sock = sckt_setup(host,server_socket);
      if (opt)
         if (write(sock, l_msg, strlen(l_msg)) < 0)          /*  send it */
         {   perror("v:  writing on stream socket");
             exit(2);
         }
   }
   else if ((opt == BACKGROUND) || (opt == GET_RESULTS))
   {
      s_msg[1] = opt + '0';                        /* build short message */
      sock = sckt_setup(host,server_socket);
      if (write(sock, s_msg, strlen(s_msg)) < 0)          /* send reply */
      {  perror("v:  writing on stream socket");
         exit(2);
      }
```

```
    read_details(opt,v->n,host,&sock);  /* receive file from fileserver */
    if (opt == BACKGROUND)
    {
      sock = sckt_setup(host,server_socket);  /* return to vote server */
      s_msg[1] = CHECKIN+ '0';
      identify(s_msg,v,&sock);
      flag = TRUE;
    }
  }
  else if (opt == COMMENT)
  {
    s_msg[1] = COMMENT + '0';                          /* build short message */
    sock = sckt_setup(host,server_socket);
    if (write(sock, s_msg, strlen(s_msg)) < 0)         /* send reply */
    {  perror("v:  writing on stream socket");
      exit(2);
    }
    bzero(buf,sizeof(buf));                            /* clean slate */
    if (read(sock, buf, 1024) < 0)
    {   perror("v:  reading message");
      exit(2);
    }
    close(sock);                                /* close socket to vote server */
                                                  /* open socket to file server  */
    fs_socket = atoi(buf);                      /* insert file server id       */
    sock2 = sckt_setup(host,fs_socket);

    strcpy(filename, "k");
    strcat(filename, argv[IDpos]);
    strcat(filename,".comment");
                          /* tell fileserver where attach this comment and */
                          /*    where to find this topic's vote server     */
    sprintf(buf,"%c %d %s %s",
            COMMENT + '0', server_socket, host, filename);
    if (write(sock2, buf, 1024) < 0)
    {  perror("v:  writing #1 on stream socket to file server");
      exit(2);
    }
    send_file(&sock2, n, filename);
    close(sock2);
    sock = sckt_setup(host,server_socket);             /* return to vote server */
    s_msg[1] = CHECKIN+ '0';                            /* build short message */
    identify(s_msg,v,&sock);
    flag = TRUE;
  }     /*  end of pro/con comment option   */

  else if (opt == ERASE)                    /* creator is using voter interface */
  {  s_msg[0] = 'A';                              /* build short message */
    s_msg[1] = ERASE + '0';
```

```
        sock = sckt_setup(host, server_socket);
        if (write(sock, s_msg, strlen(s_msg)) < 0)
        {  perror("v:  writing on stream socket");
           exit(2);
        }
        bzero(buf,1024);                    /* clean slate for next communication */
        if (read(sock, buf, 1024) < 0)
        {   perror("v:  reading message");
            exit(2);
        }
        printf("%s",buf);                   /* print servers return message */
      }
      else if (opt == LEAVE)
        printf("\n\n          Exiting to operating system.\n\n");
    } while (flag);
    close(sock);
}                   /*  end of main  */

/*****************************************************************************/
/******** Nancy Calboun, programmer ***************************************/
/************************************************** September, 1988 ******/
/*****************************************************************************/
```

PREFER:

Small Group Decision Support System Tool

by

NANCY JANE CALHOUN

A.A., Graceland College, 1962
B.S., Colorado State University, 1964
B.S., Kansas State University, 1982

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

A Group Decision Support System uses computers to aid groups in the decision process. C. Hagmann's doctoral work was the development of a Small Group Decision Support System (SGDSS). Her SGDSS included an algorithm for extracting a group preference ordering from individual orderings of alternatives.

This report describes an implementation of a distributed voting system, PREFER, which uses Hagmann's fuzzy binary algorithm. Some general background material regarding group decisions is included, followed by an overview of Hagmann's development of the algorithm.

This implementation develops voting objects which are designed to reside as daemons in machines on a local-area network. Included in the report are data-flow diagrams of the PREFER voting system, a chart showing how the modules of the system are distributed across the LAN, and examples of screen displays created by the software. One large example follows the creation of a specific PREFER voting object, its use, and the final results obtained from applying the algorithm to a completed voting session. Source code for all of the programs is included in Appendices of the report.

This implementation was tested in a LAN environment at Kansas State University. The LAN is connected with Ethernet hardware, has Unix operating systems on all machines, and uses TCP/IP software protocol. The machines include a DEC Vax 11/780, a Harris HCX-9, several AT&T 3B15's, and several AT&T 3B2's.

The report concludes that PREFER could be very useful in assisting the decision process of a small group. It could also be used in a research environment to see how computerization might change group dynamics.