HONEYWELL

MULTICS SIMPLIFIED COMPUTING AND FILING FACILITY

SOFTWARE

MULTICS SIMPLIFIED COMPUTING AND FILING FACILITY

SUBJECT

Description of the Multics Simplified Computing and Filing Facility

SPECIAL INSTRUCTIONS

This manual presupposes some basic knowledge of the Multics system provided by the 2-volume set, *Users' Introduction to Multics – Part 1,* (Order No. CH24) and Part II, (Order No. CH25).

SOFTWARE SUPPORTED

Multics Software Release 10.2

ORDER NUMBER

GL71-00

February 1984

Honeywell

PREFACE

The software product identified as the Multics Simplified Computing and Filing Facility is the property of the Massachusetts Institute of Technology and distributed by Honeywell Information Systems under license from the Massachusetts Institute of Technology.

This book is a detailed description of the Multics Simplified Computing and Filing Facility. It is intended for all users; both those who have relatively little experience on the Multics operating system (or any other computer system) and experienced programmers will find this a complete description. You are, however, expected to be familiar with the Multics concepts described in the 2-volume set, *New Users' Introduction to Multics* -- Part I (Order No. CH24), and -- Part II (Order No. CH25), referred to in this manual as New Users' Introduction.

This manual is composed of two parts: Part I (Sections 1 through 12) is a how-to-use description of the Multics Simplified Filing Facility; Part II (Sections 13 through 20) is a description of the Multics Spreadsheet Program.

Appendix A explains the procedures for using card files as input to your spreadsheet.

Appendix B is an alphabetic summary of all of the Filing Facility commands.

Manual Conventions

A few conventions and special symbols used in this manual should be introduced before you begin to explore the Multics Simplified Computing and Filing Facility.

Throughout the manual, italics is used to denote the variable portions of command lines; underlined portions of command lines can be entered instead of the entire command line.

File No.: 1U53 GL71-00

A convention used within examples is the use of an exclamation point (!) to indicate lines you type. The exclamation point does not appear on your terminal--you do not type it, and Multics does not type it to prompt you. Exclamation points appear only in examples, and only to show which lines you type. Remember too, that every line you type must end with a carriage return (RETURN).

Each section/appendix of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

LEVEL	HEADING FORMAT
1 (highest)	**ALL CAPITAL LETTERS, BOLD TYPE FACE**
2	**Initial Capital Letters, Bold Type Face**
3	*ALL CAPITAL LETTERS, ITALICS TYPE FACE*
4	*Initial Capital Letters, Italics Type Face*

CONTENTS

Tables

PART I

SECTION 1

INTRODUCTION TO THE MULTICS SIMPLIFIED FILING FACILITY

The Multics Simplified Filing Facility (MSFF) is a general-purpose filing system for small to medium amounts of data. It simulates using card decks of 3 x 5 cards. In fact, it provides a large number of blank "electronic" 3 x 5 cards on which you can enter any information you want, in any format you want, to create "card files".

Of course any filing "system" requires some organization. For example, you would expect all of the cards in a deck called "address file" to contain names and addresses and that any cards in the address file containing recipes had probably been misfiled. Similarly, you would expect only one entry, such as one name and address, on a card. And, finally, you would expect that some format would be established so that you would know what part of the information was the name and what part was the address. In some cases, you might want to get down to the street, city, state and zip code level.

If all of this seems natural to you, then using the Multics Simplified Filing Facility should also be natural. It lets you:

- Call a card file by any name you want (up to 30 characters), i.e.

  ```
  address book
  parts list
  maintenance records
  ```

- Define the items on each card and their order. You can put up to 40 items on a card and call them anything you want as long as the item name does not exceed 30 characters, counting spaces.

- Enter information with minimal regard for formatting. The Multics Simplified Filing Facility assumes one line for most items, but you can also define and use multiple line items. There is a limit of 2000 characters of information per card, about what you could squeeze onto a 3 x 5 card.

Once you have created a card file (or deck, if you prefer), there are a number of useful things you can do:

- Sort it in order of any of the items.
- Look up a card or cards that meets some matching condition, such as "name contains Jones".
- Select a subset of the cards and use or save that subset.
- Perform simple arithmetic such as count the number of cards in a group or sum an item on the cards by group, where you define what constitutes a group.
- Perform more complex arithmetic such as producing the result of an expression that uses various constants and items on the card (for example: quantity X price X 1.05 to compute cost plus sales tax) and then saving or summing those results.
- Use your card file data in Multics Spreadsheet (MSP) calculations.
- Select cards from one deck based on some match with a subset from another deck and/or combine the information from two decks.
- Produce reports with subtotals and totals on specified fields.

And, of course, you can save any card file or subset of a card file that you create and recall it for use at a later time.

SECTION 2

ACCESSING THE MULTICS SIMPLIFIED FILING FACILITY

The Multics Simplified Filing Facility runs on the Multics system. To use it, you must be a registered user on Multics.

LINKING TO THE MULTICS SIMPLIFIED FILING FACILITY

Once you have become a Multics user and have learned how to log on to the system, you might have to make a "link" to the Multics Simplified Filing Facility. This depends on where it is located on your system. Try entering the Filing Facility, as described below, prior to entering the link command. If you are successful, you can skip the linking step.

To link to the filing system, enter the following Multics command:

```
link path
```

where path is the pathname of the location of the Multics Simplified Filing Facility on your system.

ENTERING THE MULTICS SIMPLIFIED FILING FACILITY

To enter the Multics Simplified Filing Facility, type:

```
msff (followed by a carriage return)
```

The program responds by displaying:

```
GOOD EVENING AND WELCOME TO THE
MULTICS SIMPLIFIED FILING FACILITY
YOUR PERSONAL INFORMATION MANAGER
PLEASE ENTER COMMANDS WHEN RECEIVING THE CUE >
```

LEAVING THE MULTICS SIMPLIFIED FILING FACILITY

The File Facility command "finished" exits you from the program and returns you to Multics command level. If you have a terminal available, try what has been covered so far. Remember, every command line must be followed by hitting the RETURN key. The line is not entered into the computer until you press that key.

HELP ON TAP

The Multics Simplified Filing Facility provides you with online help describing any Filing Facility command. To invoke the help facility, type:

```
> help
```

when you receive the cue ">". The Multics Simplified Filing Facility responds:

```
The Multics Simplified Filing Facility is a general
purpose filing system for small to medium files of
data. It allows you to easily set up files of 'cards'
and specify what items of information are to appear on
those cards. The file structure can later be changed
by inserting new items, deleting items, and renaming
items.  All items are free form except that a
distinction is made between single line items (such as
name in an address file) and multiple line items (such
as address in that file). Facilities are available
for saving and recalling files, sorting, selection of
particular cards based on the information contained,
combining of files, and some simple arithmetic. There
is also report generator capability.

     For a listing of command categories, type 'more'.
To return to Multics Simplified Filing Facility
command level, hit 'return'.

The general categories of Multics Simplified Filing Facility
commands are:

creating a new file
saving and opening files
entering data
examining data
changing a file structure
making corrections
finding specific cards
current & original sequences
sorting
selection of subsets
cross-referencing files
arithmetic
printing

>
```

The cue ">" indicates that you are at Filing Facility command level and can enter any Filing Facility command. You can then enter the help command with a specific topic, as follows:

```
> help create
```

The Filing Facility responds by displaying:

```
The MSFF command for creating a file is:

    create a new file

You will be asked to name this file and how many
items you expect to have represented on each card.
You will then be asked to name each item.  They will
appear on the card in the order named.
 Type 'more' for the rules on naming files and items.
Hit 'return' to return to MSFF command level.

>
```

SECTION 3
CREATING CARD FILES

A card file is created with a single command. Once the card file is created, the program prompts you to specify the number of items you intend to define on each card and the names of those items. After file creation and definition, you can begin to enter data in response to the program prompts.

CREATING A PHONE BOOK FILE

The first thing you must do to create a card file is determine what data you are going to store in that file and how you might use it. The Multics Simplified Filing Facility does allow you to add and remove items from a file after you have defined it.

In the "phone book" file example below, all you want to store is a name and a phone number. Presumably, when you look up a number you want to find the name first. This implies that the names are filed in some order, but since the Multics Simplified Filing Facility lets you look up a name directly, this might not be necessary.

If possible, it would be helpful to sit at a terminal and actually do the examples as shown while reading this section.

CREATE COMMAND

After invoking the Multics Simplified Filing Facility, enter the following create command in response to the Filing Facility command level cue ">". The create command initiates a dialogue that lets you name the new file, specify the number of items on each card, and supply the name for each item (item name) on the card. In the card file creation session illustrated below, your input is indicated by an exclamation point (!) for clarity only and is not required as part of the input.

The Multics Simplified Filing Facility considers the space between words in names as part of the name. For example, using two spaces between a file name would (as far as the Filing Facility is concerned) change the name. The Filing Facility also distinguishes between upper case and lower case letters.

```
> create a new file

Give me the name of this new file: !phone book (and hit RETURN)
How many items do you plan to have on each card? !2
Give me the name of the first item: !name
Give me the name of the second item: !phone number

> (indicating that it is waiting for the next command.)
```

If you wish, you can now type:

```
> display the active file information
```

and the program displays what it knows about the file called phone book.

ENTERING DATA

Now that you have created the phone book file, you can begin entering some data. To enter a card, type the following enter command:

```
enter a card
```

To enter more than one card, you can enter:

```
enter n cards
```

Type the command "enter 2 cards". (Note that numbers in command lines are always entered as digits.) If you hit the RETURN key after entering that line, the program responds:

```
name:
```

indicating that it wants the data "name" for the first card. On that same line, you could type:

```
Doe, Jane L.
```

and it responds:

```
phone number:
```

You enter:

```
253-4444
```

and it responds:

```
name:
```

indicating that it wants that data for the second card, and so on.

SAVING AND OPENING FILES

After you have entered the data for the last card, in this case the second, the program responds with the cue ">" indicating that it is waiting for the next command. For the moment, save this file and leave the Filing Facility. To save the information you entered, you must type the following command:

```
> save this file
```

and when the program responds with the cue, you can type:

```
> finished
```

which returns you to Multics command level.

Now go back into the Multics Simplified Filing Facility by typing:

```
msff
```

The Filing Facility displays:

```
GOOD EVENING AND WELCOME TO
THE MULTICS SIMPLIFIED FILING FACILITY
YOUR PERSONAL INFORMATION MANAGER
PLEASE ENTER COMMANDS WHEN RECEIVING THE CUE >.
```

To open the phone book card file, enter the following open command:

```
> open the file phone book
```

The Filing Facility displays:

```
2 CARDS IN THE phone book FILE
```

To view all of the cards in the file, enter:

```
> show me all cards
```

The Filing Facility then displays all of the cards in the file:

```
+ card 1  ------------------

|name:              Doe, Jane.
|phone number:      253-4444

+ card 2  ------------------
|name:              Roe, Mary
|phone number:      333-3333
```

(This above display shows the actual form of the output. Henceforth, the headers are omitted for clarity.) This demonstrates that you have saved the file and are able to retrieve it. It also shows you one way of looking at the data.

CREATING A PARTS LIST FILE

This example is a bit more complex. Assume here that you want to store the part number, the part name, the material, the primary vendor, a secondary source, and the number usually ordered.

To create a "Part's List" card file, invoke the Filing Facility and then enter the following create command:

```
        > create a new file
```

The Filing Facility begins to prompt you for the pertinent information about your new Part's List card file as shown below. Note that input you must supply is preceded by an exclamation point (!) for clarity only and is not required as part of the input.

```
        Give me the name of this new file:  !parts list
        How many items do you plan to have on each card?  !6
        Give me the name of the first item:  !part number
        Give me the name of the second item:  !part name
        Give me the name of the third ite:  !material
        Give me the name of the fourth item:  !primary vendor*
        Give me the name of the fifth item:  !secondary vendor*
        Give me the name of the sixth item:  !number ordered
```

Note that the use of the asterisk (*) is explained under "Multiple Line Items" below.

Once you have created the file as described above, you can begin to enter information on the cards defined. To enter information for 10 cards in the card file, type the following enter command:

```
        > enter 10 cards
```

The Filing Facility begins to prompt you for the information required for each of the items that you defined on the card. It continues to prompt you, as follows, until you have filled 10 cards. Note that information you must supply is preceded by an exclamation point (!) for clarity only and is not required as part of the input.

```
part number:          !b123
part name:            !pivot bearing
material:             !case hardened steel Rockwell 49C
primary vendor:       !Jones Bearings
                      !123 Main St.
                      !Town, State*
secondary vendor:     !Aunt Sally's Parts
                      !Lilac Lane
                      !Town, State*
number ordered:       !12,342
part number:          !b137
```

PERUSING A CARD FILE

There are a number of commands that you can use to peruse a file. Since they are all quite simple and self-explanatory, they are simply listed here. (Note that throughout this manual you can enter the entire command line as shown, or you can enter only the underlined portion of the command to achieve the same results.)

```
show me this card
show me the previous card
show me the next card
show me the first card
show me the last card
show me all cards
show me the next n cards
show me the previous n cards
skip to the first card
skip to the last card
skip forward n cards
skip back n cards
go to card n
```

You can enter these commands exactly as shown (without the underlining), except that a number (digits only) replaces the n; or you can enter only the underlined portions. In other words, "previous 4" and "show me the previous 4 cards" are equivalent. If you omit the n, 1 is assumed. Enter a few more cards into your file and try them out.

MULTIPLE LINE ITEMS

In our phone book file example, the items used could be entered on one line. If you enter an item that takes more than one line, you have to follow certain conventions.

First, when you initially define the name of a multilined item, you must end the item name with an asterisk (*). (You can forever after ignore that asterisk in the item name, although it is printed when entering or displaying a card as a reminder that it is a multilined item.)

Second, when entering the data for that multilined item, you must end it with an asterisk. In the following example, an "address book" file is created and the card has 1 multilined item defined: address. Note that an asterisk is required when you are finished entering the last line of the address. (Note that your input is preceded by an exclamation point (!) for clarity only and is not required as part of the input.)

```
> !create a new file

Give me the name of this new file:  !address book
How many items do you plan to have on each card?  !2
Give me the name of the first item:  !name
Give me the name of the second item:  !address*
```

Once the card format is defined, you can begin to enter information in the card file:

```
> !enter 2 cards
```

The Filing Facility then prompts you to enter the information for 2 cards:

```
name:       !John Smith
address*:   !123 Main St.
            !Anytown,   USA*
name:       !Jack Jones
address*:   !Apt. C
            !123 Woodard Ave.
            !New York, NY*
```

CHANGING AND CORRECTING DATA

Since data does change and we all make mistakes, there has to be a way to change and correct data after it is been entered. The Multics Simplified Filing Facility provides two facilities, one for changing the entire item and one for changing only a part of the item, as described below.

Changing An Entire Item

Assume that Jane Doe changed her phone number; change that item by first selecting the card to be changed (in this case, the last card entered):

```
> !show me the last card
```

The Filing Facility displays:

```
name         :  Doe, Jane
phone number:  253-4444
```

To change the old phone number to the new phone number, type the change command, as follows:

```
> change phone number to 235-5555
```

Note that italics is used to denote the variable portion of a command line.

```
phone number:   235-5555

>
```

Changing Part of a Multilined Item

Now, assume that Jack Jones moved to a different apartment in the same building and you only want to change that part of his address:

```
> show me the first card
```

The Filing Facility displays:

```
address*:   Apt. C
            123 Woodard Ave.
            New York, NY
```

To change the address, enter the change command as follows:

```
> change addresss Apt. C to Apt. F
```

The Filing Facility displays:

```
name:       Jack Jones
address*:   Apt. F
            123 Woodard Ave.
            New York, NY

>
```

By including the characters to be changed after the item name, you indicate that you only want to change that portion of the item. Obviously the word "to" in this command is a key word separating the old string from the new. Just as obviously, the old string cannot contain the word "to".

When changing a multiple line item, the change must be included in its entirety on one line. You can use an asterisk (*) to designate the desired end of a line. For example:

```
change address to Apt G* 275 Lincoln Street* New York, NY.
```

If the new item does not fit on a single line, use a special character as a marker as follows:

```
change address to fourscore and seven years ago, our forefathers +

change address + to met upon this continent to establish +

change address + to a new nation etc.
```

If this method becomes too cumbersome, especially with the program printing each change, you can delete the old card and insert a new one.

You should note that although changes appear to have taken effect immediately, actually both versions, the changed and the unchanged, are known to the program and the changes are not permanently made to the file until you evoke the command "save this file" and answer "yes" to the question DO YOU WANT TO INCORPORATE THE CHANGES YOU HAVE MADE? In a later section, the commands for saving subsets and retaining subsets are presented.

Making Global Changes

Occasionally, you might want to make the same change to all of the cards in the subset. You do this by using the prefix "globally " before the change command. The syntax is:

```
globally change item_name string to new string
```

For example:

```
globally change dept to Ocean Engineering
```

This would change the contents of department name in all of the cards in the current subset. If you only wanted to change those cards where the field contained a specific string, you still use the prefix, but you use the second form of the change command. For example:

```
globally change room 39-530 to 39-430
```

or, using our previous example, but assuming many departments are included in the file:

```
globally change dept Naval Architect to Ocean Engineering
```

In this case, as well as in the case of changing values on single cards, the change does not become a permanent change in the file until you save the file or subset, and then only if you specify that you want the changes saved.

Adding, Removing, or Changing an Item

If after you have set up your card file you find that you should have included another item, you can add it to the file using the command:

```
insert an item *item name1* before *item name2*
```

This places a new item (an empty item) with the name supplied as *item name1* on each card in the file before the item named as *item name2*. If you want this new item at the end, use "the end" in place of *item name2*. For example:

```
insert an item *max price* before *number ordered*
insert an item *min inventory* before the end
```

As mentioned, this new item is empty. It can be filled in by using the change command, the fill in command (described below), or the set command, which is described in another section.

You can also remove items from a file using the command:

```
remove the item *item name*
```

Finally, you can also change the name of any item in case you decide you no longer like your first choice. The command is:

```
change the name of *item name* to *item name*
```

The Fill In Command

After inserting a new item on each card in the card file, you would then want to "fill in" the information for the new item on each card. The "fill in" command lets you fill in the information for the new item on as many cards as you choose.

To fill in only the current card, enter:

```
fill in *item name*
```

To fill in the new item on more than one card, enter:

```
fill in *item name* on *n* cards
```

where *n* must be a digit and *item name* items in your active file. This processes the next n cards starting from your current card.

You can also "key on" any other item or items on the card. The item name(s) keyed are displayed along with the prompt for the new item. To create a key for the fill command, use:

```
fill in *item name* on *n* cards keying
on *item name(s)*
```

(Note that multiple item names must be separated by commas.) This processes n cards, starting with the present card, and prints out the names of the items and the data in the items you are "keying on" (there can be up to six). The program then prints the name of the item to be filled and waits for your input.

You can leave out "on n cards" if you intend to fill in only the present card. You can also use the word "all" instead of a digit, such as in "on all cards", if that is what you intend.

If you leave out the phrase "keying on" and the subsequent item names, no information is printed about the card you are filling in. This can be correct if you are only filling in the present card, but it is not recommended that you "fly blind" when filling in several cards.

For example, if you wanted to fill in information for the new item you previously inserted on each card in the card file, you would issue the following command:

```
> fill in occupation on 2 cards keying on name
```

The name associated with the card is displayed and you are prompted to enter the data for occupation, as follows (note that your input is preceded by an exclamation point (!) for clarity and is not required as part of the input):

```
name                :AHLIN, DR. JEFFERY
occupation          :!dentist

name                :ARMIDA'S
occupation          :!restaurant

>
```

You are then returned to the Filing Facility command level.

IF THEN ELSE COMMAND

The "if then else" command adds flexibility to your Filing Facility programs by allowing you to add "decision making" to them. The syntax is:

```
if conditional expression then action
else action
```

where *conditional expression* is any of the conditional phrases that can be used in a selection and *action* is one of the following actions:

```
delete this card

let item name = expression

change item name to string
```

For example, to change one item based on the value of another:

```
> if amount >13,000 then let discount = .12*amount
                    else let discount = .08*amount
```

You can also omit the "else" clause. For example, to delete all of the cards in a card file that contain "Boston" in the item name "City" you would enter:

```
> if City includes Boston then delete this card
```

The "if then else" command searches the card deck checking each card against the conditional. If the condition is met, then the action specified in the "then" is taken for that card. Otherwise the action after the "else" clause is taken. If you wish, you can omit the "else" clause and use only:

if *conditional expression* then *action*

If you have no 'then' action, as in:

```
> if City includes Boston then else delete this card
```

the word "then" must appear just before the word "else". In other words, it cannot be missing; it is a necessary key word.

Notice that the actions are single card commands. Each card is examined and then the appropriate action is taken for that card. For example, you cannot say:

```
>  if City includes Boston then delete these cards
```

The "if then else" command can be very useful. For example, you could not pick the subset of cards where City does not include Boston using the select command. The condition "does not include" is not available. The example given above, however, accomplishes this.

```
> if account is 13601 then change parent to 13600
```

If you cannot do what you want with the other Multics Simplified Filing Facility commands, then you should be able to do it using this command else it cannot be done.

COMMANDS INTRODUCED IN THIS SECTION

As before, you can enter the commands as shown, where a short form is indicated, by entering just the underlined words. Note that multiple item names must be separated by commas.

change the name of *item name* to *new item name*
change *item name* to *string*
change *item_name string* to *new string*
create a new file
display the active file information
enter a card
enter *n* cards
fill in *item name*
fill in *item name* on *n* cards
fill in *item name* on *n* cards keying
  on *item name(s)*.
globally change *item name* to *string*
globally change *item name string* to *new string*
go to card *n*
if *conditional expression* then *action*
  else *action*
insert an item *item name* before *item name*
open the file *file name*
remove the item *item name*
save this file
show me all cards
show me this card
show me the previous card
show me the next card
show me the first card
show me the last card
show me the next *n* cards
show me the previous *n* cards
skip to the first card
skip to the last card
skip forward *n* cards
skip back *n* cards

SECTION 4

REORDERING A CARD FILE

Card files have two sets of indices that keep track of the sequence of the file: the original sequence and the current sequence. The original sequence of a card file is usually the order in which the cards were entered. This is also the "current sequence", unless and until you do something to change it. There are five ways you can change the current sequence. These are by moving cards, by sorting the file, by inserting cards, by deleting cards, and by copying cards. created called the "current subset"

MOVING CARDS

The move command changes the current sequence by making the move indicated. The various forms of this command are:

```
move this card before (or after) card n
move card n before (or after) this card (or card n)
```

where n is replaced by a card number.

SORTING THE CARD FILE

You can sort a card file using any of the following methods:

- Alphabetic sort
- Sort by value
- Sort by date
- Sort by table
- Sort on several fields

Each method is described below.

Alphabetic Sort

The basic sort command sorts a file in alphabetical order based on the item name you specify. The format for the sort command is:

sort on *item name*

The sort command rearranges the current sequence of the card file so that it appears to be sorted in alphabetic order on the specified item. In fact, however, only the current sequence has been changed. To permanently save the card file resequenced by the sort command, you have to issue the reset command (see Restoring and Resetting Card Sequence).

Sort by Value

A field that is sorted by numeric value (numeric sequence) can contain letters; the symbols plus (+), minus (–), and decimal point (.); or digits, but only the digits and signs have any effect on the numeric value of the field generated; the rest are ignored.

The syntax for the command is:

```
sort by value on item name
```

Note that if you sort a numeric field alphabetically, the number 123 appears before the number 9 since the numbers 1,2, and 3 are each individually less than 9.

Sort by Date

A field that is to be sorted on date can be of the form: mm/dd/yyyy (01/01/1985) or month_name,day,year (January 1, 1985). You can use the full name of the month or the first three letters of the month. Specify the year as four digits or only the last two digits (no apostrophe as in '84), and the day must be followed by a comma.

The syntax is of the command is:

```
sort by date on item name
```

Sort by Table

Sorting by table gives you complete control over the sequence of the file. The command:

```
sort by table on item name
```

lets you define a set sequence for a particular item. For example, if you had an item on a card called Faculty Rank, the data in that item could contain any of the following titles: Department Head, Professor, Associate Professor, Assistant Professor, Lecturer, and so on. An alphabetic sort on Faculty Rank would produce: Assistant Professor, Associate Professor, Department Head, Lecturer, Professor which would be unacceptable since you want their respective rank listed in sequence. To achieve a correct sort, you have to create a "table" containing the order of preference of each possible value for Faculty Rank in the form of a Multics segment and call it table_file.name, where you specify the name. Create the file, using an editor, with the following form:

```
5
PEPL
"Department Head"
"Associate Professor"
"Assistant Professor"
"Lecturer"
```

The first line contains a number (in this case "5") specifying the number of items in the list (including the PEPL statement). This number can be as great as 100. The second line is optional, but if included can contain the PE and/or the PL statement below.

PRINT EXCEPTIONS

A Print Exception (PE) is any value that appears on any card in the field being sorted that does not appear in the table. Values that appear on the cards that do not appear in the table (exceptions) are arbitrarily assigned the next place in the table if room permits. This ensures that all Teaching Assistants appear together in the final sorted file as will all Instructors. If this control is not present, no sorting of exceptions is attempted; they simply remain in the place they fall in relation to all other exceptions.

PRINT LIST

Print List (PL) specifies that the list of items in the table are to be displayed.

If either PE or PL is missing from the second line, the corresponding action is not taken. If this control card does not appear at all and the second card is simply one of the values in the table, neither action is taken.

When you invoke "sort by table", the program asks you for the name of the table you want to use. Reply with only the last part of the table segment. For example, if you generated a file called table_file.rank for our example, you would reply "rank" when asked for the table name. You can have as many tables as necessary to handle your special cases.

Sorting on Several Fields

To sort two fields, one within the other, requires two sort commands: first on the inner item, then on the outer. Likewise, a three field sort requires three sorts, etc.

INSERTING AND DELETING CARDS

The insert command has the following format:

```
insert a card
```

Inserting a card "appears" to place that card before the one you are currently viewing rather than at the end of the file as "enter a card" does. However, only the current sequence has been changed. In fact, the original sequence does not even include cards which have been inserted. This is not true with cards that are "entered".

You can also delete a card from the current sequence using the command:

```
    delete this card
```

Again, the deleted card is still in the file and is known to the original sequence. You can, however, reset the original sequence so that it removes this card, as is explained in "Restoring and Resetting Card Sequence".

COPYING CARDS

The request to copy cards adds duplicates of the card being copied directly after that card in the current subset. The various forms of the request are:

```
    copy
```

which makes one copy of this card, and

```
    copy this card n times
```

where *n* is a number or any legitimate arithmetic expression. Each card is copied the number of times indicated by subtracting one from the value of the item quantity on that card. (See Section 7 for a further explanation of expressions.)

```
    copy all cards n times
```

which copies all of the cards in the file the required number of times.

```
    copy... n times changing item name to value
```

for each card copied the specified item is changed to the specified value.

```
    copy... n times setting item name to n [converted to...]
```

for each card copied, the item is set to the arithmetic value of the expression and converted if specified (the set command is described in Section 7).

This request changes the current sequence.

RESTORING AND RESETTING CARD SEQUENCE

As mentioned, sorting, copying, deleting, and inserting only change the current sequence. In effect, this allows you to create temporary versions of the file for some specific purpose. (When subsetting is presented, you will see a more powerful extension of this facility.) When you want to restore the file to its original sequence, use the command:

```
    restore the sequence
```

If you insert or delete cards, the number of cards in the two sequences becomes different. If your original card sequence will lose cards by issuing a restore command, the program asks you if you want to proceed. If you say "yes", the file is restored to the original sequence and the original cards. As mentioned earlier, saving the file always saves it in the original sequence.

If you want to reset the original sequence to the current sequence, you can do this with the command:

```
reset the sequence
```

Again, if it looks like you will lose cards, you are asked if you want to continue the operation.

To create and save a new card file that has the original sequence set to the current sequence of this card file, use the command:

```
save this subset
```

You are then asked to name the new file.

To create another open file similar to the active file but with the original sequence reset to the current sequence, use the command:

```
retain this subset
```

Again, you are asked to assign a name to the new file.

The commands "save this subset" and "retain this subset" are used to save or open selected subsets of the data that is described in the next section.

COMMANDS INTRODUCED IN THIS SECTION

The following commands for reordering a card file were presented in this section. Note that words in italics indicate variable portion of the command line and underlined words can be entered instead of the entire command line.

```
copy this card n times
copy all cards n times
copy item name changing item name to value
copy item name setting item name to expression
delete this card
insert a card
move this card before (or after) card n
move card n before (or after) this card (or card n)
restore the sequence
reset the sequence
retain this subset
sort on item name
sort by value on item name
sort by date on item name
sort by table on item name
save this subset
```

SECTION 5

INFORMATION SELECTION AND RETRIEVAL

One of the most important features of the Multics simplified filing system is the facility it provides for "finding" or "selecting" cards that meet your specified criteria. To find cards or select cards, use either the find command or the select command, respectively. Both of these commands are flexible and extensible enough to meet most selection or retrieval requirement.

THE FIND COMMAND

When you "find" a card or cards, those cards are displayed and the current sequence of the card file is not changed.

The basic format for the find command is:

```
find the card where conditional expression
```

As a very simple example, if the file "phone book" was open and you wanted to look up Jane Doe's phone number, you could say:

```
> find the card where name is Doe, Jane L.

name:          Doe, Jane L.
phone number:  235-5555
```

Additional find commands are:

```
find those cards where conditional expression

find the next card where conditional expression

find the next n cards where conditional expression
```

These commands do not change the current sequence, they simply display the cards selected, if any.

THE CONDITIONAL EXPRESSION

The conditional expression (or conditional, for short) specifies exactly the conditions to be met in finding or selecting a card. Its form is:

```
item_name condition value
```

where:

item name is the name of the item name, such as "phone number", to be used in the match

condition is one of the phrases:

- for alphabetic comparisons

  ```
  - is or comes before
  - is or comes after
  - is
  - is not
  - comes before
  - comes after
  - starts with
  - contains  (searches for an approximate match)
  - includes  (searches for an exact match)
  ```

- for numeric comparisons

  ```
  - >   ("is greater than")
  - >=  ("greater than or equal")
  - ^=  ("not equal to")
  - =   ("is equal to")
  - <=  ("less than or equal")
  - <   ("is less than")
  ```

- for date comparisons

  ```
  - is on or earlier than
  - is on or later than
  - is on
  - is not on
  - is later than
  - is earlier than
  ```

value is the value being used in the comparison.

The following are examples of conditionals:

```
name is Doe, Jane L.
price > 100
> name starts with D
date is later than Jan 12, 1983
name includes Jane
```

Complex Conditional Expressions

You can join conditionals to make complex conditionals using the conjunctions "and" and "or".

and	joins two conditionals where each must be met before that card is selected
or	joins two conditionals where if either condition is met that card is selected

Since "and" and "or" are keywords in the conditional expression, if they appear in a value, i.e. "Laurel and Hardy", you should precede them by a tilde, thus: Laurel ~and Hardy.

An example of a complex conditional would be:

```
> name starts with D and name contains Jane
price > 100 or quantity > 10000 and price > 50
```

Multiple Choices

You might want to specify several values that match an item to meet your criteria. You could do this using the "or" conjunctor as follows:

```
where object code is 320 or object code is 330 or ...
```

A simpler procedure to use is the vertical bar character, "|", as follows:

```
where object code is 320|330|....
```

The only way to specify an empty field is with the vertical bar. For example:

```
where eft is |
```

translates into where eft is "empty", or where eft is "empty". Whereas,

```
where eft is |1.0
```

translates into where *eft* is "empty" or is "1.0".

THE SELECT COMMAND

When you "select" a card or cards, those cards are <u>not</u> displayed but are retained in a subset defined by the current sequence.

The select command lets you select a subset of the cards currently contained in the current sequence. (Remember when you first open a file this is all of the cards.)

After the select command is executed, only those cards that have been selected are included in the current sequence, and all operations are performed on that subset of cards until you do a "restore the sequence".

The form of the select command is similar to the find command. It is:

```
select those cards where conditional
```

The conditions are also the same as for the find command. The difference is that the find command displays the card(s) but does not change the current sequence. The select command does not display any of the cards but retains them in the subset defined by the current sequence. An example of a select command might be:

```
> select those cards where primary vendor starts with Aunt Sally
```

THE INDEX COMMAND

The index command is similar to the select command; it causes the subset of cards that match the conditional to be chosen. Additionally, it produces an index of those cards by listing the specified field or fields. For example:

```
> index of name where address contains New York, NY
```

The Filing Facility then displays the names of all those who have "New York" specified in their address, as follows:

```
0001 Koch, Edward
0002 Smith, John
0003 Brenner, George
0004 Wilson Industries
```

If you wanted other items listed in the index, the syntax is:

```
index of item name(s) where conditional
```

Note that multiple item names must be separated by commas.

Up to five items can be listed. For duplicate lines, only one line is printed, thus you can get many fewer lines than you have cards in the set. In this case, the card number printed is the number of the first card having that configuration.

If no "where *conditional*" appears in the command line, the index command provides an index listing of the current subset. In many cases, this can be preferable to the "show me" commands. While the index command is listing your file, you can hit the BREAK key at any time. This brings you back to the program's command level. You can use this feature to do a quick scan and look for a particular item.

The command "go to card n", where n is the card number, positions you at that card. If you hit the BREAK key too late, you might return to Multics command level. Typing "pi" returns you to the Multics Simplified Filing Facility.

SAVING A SUBSET

On occasion, you might want to retain a subset for later use during a session and then restore the original file. Do this with the command "retain this subset" which, in effect, opens another file whose original and current sequence are the current sequence of the subset. In other words, this new file is the subset. You can then go back to the original file, using the command "change to file *file name*" and restore the sequence.

Another way to save a subset is the command "save this subset". This actually stores away a copy of the subset as a new file.

COMMANDS INTRODUCED IN THIS SECTION

The following selection and retrieval commands were introduced in this section:

```
change to file file name
find the card where conditional expression
find those cards where conditional expression
find the next card where conditional expression
find the next n cards where conditional expression
index of item name
index of item name where item name contains name
select those cards where conditional
```

SECTION 6

OPEN AND ACTIVE CARD FILES

In the preceding section, it is implied that you can have more than one file open at a time. In fact, you can have up to 30 files open at any time, though only one file is the active file. Open files can be combined and cross-referenced to create entirely new files, if required.

OPEN CARD FILES

To determine which files are open, use the command:

```
display the open file information
```

or its short form "display open". This command tells you which files are open and which is the active file.

The command:

```
display the active file information
```

or "display active", which was mentioned earlier, tells you the name of the active file, the number of items and their names, and the number of cards in the original and current sequences.

To change from one active file to another, use the command:

```
change to file file name
```

where *file name* is replaced by the file name, or the short form "file *file name*".

CROSS-REFERENCING AND COMBINING CARD FILES

Although having several files open at once and being able to jump from one to another might be convenient, the benefit comes from the ability to cross-reference and combine files. There are five commands to do this:

```
couple with file name
choose from file name on item name(s)
combine with file name on item name(s)
compose with file name on item name(s)
difference from file name on item name(s)
```

Their function and use are described below. Note that multiple item names must be separated by commas.

Coupling Card Files

The following command lets you join two files of similar structure:

```
couple with file name
```

Thus, if files a_file and b_file both have the same items in the same sequence and a_file is the active file, the command "couple with b_file" results in the cards in the current subset of b_file being added to the current subset of a_file. In this case, b_file is left unchanged, though it should be noted that any changes made to the b_file cards now incorporated in a_file is reflected in that card in b_file. In other words, the couple command simply links the files; it does not create any new cards.

The couple command is normally used when you are building up a file through a series of selects and couples from another file or when you have several similar small working files, such as one for each department, that you want to join for some overall report.

Choosing - A Different Form Of Select

The following choose command lets you select cards from one file that match cards from another file in specified fields (in each case it is the current subset that is used):

```
choose from file name on item name(s)
```

For example, the file Books has the items title, author, publisher; and the file Publishers has the items publisher and address. You could select all of the cards from the file Books that contain certain keywords in their titles and then use "choose" to find the publishers' address cards. For example:

```
> change to file Books

> select those cards where title contains Petroleum | Gasoline

8 CARDS SELECTED

> choose from Publishers on publisher

3 CARDS SELECTED

> all  (to print the cards selected)
```

Note that the choose command does not create a new file; the current subset of the named file is changed to include only the chosen cards.

Note that although it is expected that the fields used for matching have the same name in both files, it is not necessary. If, for example, the Publishers file had the items company name and address rather than publisher and address, when you enter the command "choose from Publishers on publisher" the Multics Simplified Filing Facility replies that the item publisher does not exist in the file Publishers and asks for the "correct" name. You would then reply "company name", and the selection would proceed. Note, however, selections are made only if the values are identical.

Another example that might occur to you is to find all the books written by the authors included in the selected set. In other words, you might want to match the active file against the active file. Since you want to match the current subset against the original subset, however, you have to issue a "restore the sequence", and that would lose your subset. The answer, of course, is to "retain this subset", "change to file Books", "restore the sequence", "change to file (whatever you named the subset)" and finally, "choose from Books on author". A bit awkward, but it can be done.

You might want to select on more than one field. For example, if you wanted to pull all the green Chevrolet sedan and blue Ford sedans out of a list of motor vehicles, you could create a file of two cards having the items Make, Model and Color, insert the correct data, and then type:

choose from *motor vehicles* on *Make,Model,Color*

It is important to know how the choose command actually proceeds. First, it does an alphabetic sort on both files on the fields being used for the selection. It then matches the first card in each deck. If they match, the card from the named deck is selected and moved aside and the next card in that deck is matched against the first card in the active deck. If they do not match, the lower in the sequence is moved and the comparison is made with the next card. In the example above, the first card in the active file Chevrolet, sedan, green would be the card matched against until the last of the Chevrolet, sedan, green cards in the motor vehicles file had passed. After that, all cards would be matched against Ford, sedan, blue until the last blue Ford sedan was selected. At that point, the selection would be completed. Note that if multilined items are used in the match, there is no guarantee of a proper selection.

Difference - The Opposite Of Choosing

The following difference command is similar to "choose from" except that the cards in the named file, which do not have a match in the active file, are the ones chosen:

difference from *file name* on *item name*

Combining Card Files

The combine command takes two dissimilar files, that is files that do not have the same items, and produces a third having all the cards that matched in the specified field and all the items contained in the two files. This third file then becomes the active file. The format is:

combine with *file name* on *item name(s)*

In the case where items in both files have the same name, only the item from the first (the active) file is included in the new file. All items in the second (the named) file that do not match, by name, items in the first file are included even if they are used to make the selection. Thus, if one file has the item Title and the other Rank and you match on that item (see "Choosing - A Different Form of Select"), both Title and Rank appear in the third file though their values are identical.

Composing Card Files

The compose command is similar to the "combine with" command except that the information from all of the cards in each file is retained. Where cards from the two files match in the specified fields, their common information is combined onto one card, as with the combine command. A card in one file that does not have a match in the other file is still retained; however the missing items are left blank.

The syntax of the compose command is:

```
compose with file name on item name
```

The compose command, unlike the combine command, does not allow a "one to many" matching; once a card is matched (or it is determined that it does not match) the information is copied into the new file and the program moves to the next card.

The compose command also can be used as a more sophisticated version of the couple command where two files are not required to be identical. Additionally, the compose command merges two files rather than simply linking them.

As in using the combine command, the compose command asks you to supply a new file name and the specifed new file name becomes the active file.

COMMANDS INTRODUCED IN THIS SECTION

The following file information commands were introduced in this section:

```
change to file file name
choose from file name on item name(s)
combine with file name on item name(s)
couple with file name
difference from file name on item name
difference from file name on item name
display the open file information
display the active file information
```

SECTION 7

NUMERICAL CALCULATIONS

The Multics Simplified Filing Facility allows a general capability for doing numerical calculations. The commands and their various forms are described below.

THE COMMANDS COUNT, SUM, SET, LET, AND EVALUATE

The following is a list of the various forms of the count, sum, and set commands.

count
: gives a count of all the cards in the subset.

count by *item name(s)*
: gives a count for each group and subgroup defined by the list. For example: "count by school,department" would provide a count for each department, a total count for each school and a grand total count. Assuming, of course, that the file has been sorted correctly.

sum of *item name*
: totals the values in the field specified and gives a count and this total for the subset.

sum of *item name* by *item name(s)*
: totals the values in the field specified and gives subtotals and counts for each group as well as for the grand total.

sum of *expression*
: totals the value of the expression. (See below for a further definition of expressions.)

sum of *expression* by *item name(s)*
: totals the expression by groups.

set *item name* to *expression*
: sets the value of the specified item for each card in the current subset to the value of the expression for that card.

let *item name = expression*
: sets the value of the specified item for <u>this</u> card to the value of the expression.

evaluate *expression*
: prints the result of evaluating the given expression for the current card.

EXPRESSIONS

Expressions consist of operands, which are either constants or item names, and operators that tell how the operands are to be combined. The allowable operators are:

+ (plus)	adds two operands
- (minus)	subtracts the second operand from the first
* (multiply)	multiplies two operands
/ (divide)	divides the first operand by the second
** (exponentiation)	raises the first operand to the power of the second

The simplest expression is just one operand. For example:

```
> set price to 12.00

> sum of cost
```

As operands and operators are added, the expression can become complex.

```
> set price to cost + 1.10 + fixed overhead
```

Since expressions are always evaluated left to right, (that is, the first operator is applied to the first two operands and the next operator to this result and the next operand) it is sometimes necessary to group terms. This is done using the parentheses. For example, (2*3)+7 and 2*3+7 both yield 13, but 2*(3+7) results in 20 since the expression within the parentheses is treated as one operand.

DECIMAL PLACES IN RESULTS

Normally the Multics Simplified Filing Facility expresses all results to two decimal places. You can change this, however, with the following command:

```
fix decimal places to n
```

where n is an integer representing the number of decimal places you want expressed in future results. It can be any number from 0 to 12.

MULTICS SIMPLIFIED FILING SYSTEM PARAMETERS

There are currently two special parameters that can be used in expressions. They are &no_of_cards and &index. Any time &no_of_cards is used in an expression its value is the number of cards in the current subset. Using the parameter &index provides the number of the current card in the current sequence. An example of their use would be:

```
set rank to &no_of_cards - &index + 1
```

which would provide a rank in reverse order of the current sequence. (You can also use &index or &no_of_cards as simple commands to return that information.)

RETAINING SUMS AND COUNTS

You might sometimes want to produce a file that represents an aggregate of another file. For example, given a file that represents the detailed budget by department, section, and expense item you might want to generate a file which contains the sum of budgeted expenses for each section and department. You can generate such a file by using the "retain" command in conjunction with the sum command. The syntax in this case would be:

retain sum of *budgeted expense* by *department, section*

You are asked what you want to call this new file. The program then prints out the counts and sums for each department and section combination, just as it does if you invoked "sum" without the retain, but it also is producing a card in this new file that contains department, section, count and sum. You also get a department count and total in your printout as well as a grand total at the end, but there are no cards produced in the file to match these lines. When the operation is finished the new file becomes the active file.

The retain command can also be used in conjunction with the count command.

DATE FIELDS

You might occasionally want to do some simple arithmetic using fields that contain dates, such as:

set *no. of days* to *end date - start date*

This is a legitimate expression. The key that makes it legitimate, however, is that the four characters "date", "Date", or "DATE" appear somewhere in the item name. If so, the program trys first to interpret that item as a date. If it can, it gives it a value equivalent to the number of days between December 31, 1899 and that date. (Only dates between January 1, 1900 and December 31, 1999 are interpreted correctly.) If the item is not a legitimate date, then the program converts it to a number in the usual way.

There is also a parameter for use in date calculations; it is "&today". It represents the number of days between December 31, 1899 and the current date. The expression:

```
completion date - &today
```

computes the number of days to the date in the field completion date.

CONVERTING THE FORM OF DATA

To convert the form of your data, use the following syntax:

set *item name* to *expression* converted to *type* the above command changes the form of the final result according to the type. For example:

```
set day to Sep 12, 1983 + 4
```

results in day being set to 30574, which is the result of changing a date to a value (see above). The form:

```
set day to Sep 12, 1983 + 4 converted to date
```

results in day being set to September 16, 1983, however. Similarly using "converted to day of the week" would set day to "Friday", and using "converted to month" would set day to September 1983.

Those three "converted to date", "converted to day of the week" and "converted to month" assume the input is a date index (number of days since December 31, 1899). Other types of conversions using hours and minutes are:

```
"converted to hours" (minutes to hours, such as 2.5)
"converted to hours and minutes" (minutes converted to 2:30)
"converted to minutes" (hours such as 2.5 converted to 150)
```

In addition, the Multics Simplified Filing Facility can convert the linear, volume, and weight measures shown in the table below.

Linear Measures	Volume Measures	Weight Measures
inches	pint	ounces
feet	quart	pounds
yards	gallon	tons
rods	liter	long tons
chains	cubic inches	grams
miles	cubic centimeters	kilograms
nautical miles		
meters		
kilometers		
cemtimeters		

Table 7-1. Measure Conversions

The syntax used to convert these measures is:

```
... converted from measure to measure
```

An error message is displayed if you try to use a measure it does not understand or if you try to mix different types of measures.

MULTICS SIMPLIFIED FILING SYSTEM FUNCTIONS

Functions are similar to commands but functions perform some mathematical calculation for you. There are several functions built into the Multics simplified filing system. Each takes only a single argument (though each argument can be an expression) and returns a single value. Their form is:

```
&function (argument expression)
```

The functions are:

&integer	returns the integer portion of the value of the expression (truncated)
&floor	returns the highest integer that does not exceed the value of the expression &floor(2.3) = 2, &floor(-2.3) = -3
&ceiling	returns the lowest integer that is not exceeded by the value of the expression &ceiling(2.3) = 3, &ceiling(-2.3) = -2
&day	returns a number 1 through 7 corresponding to the day of the week of a date index
&day	(January 1, 1983) = 7 (Saturday)

COMMANDS INTRODUCED IN THIS SECTION

The following numerical calculation commands were introduced in this section:

```
count by item name
evaluate expression
fix decimal places to n
let item name = expression
retain _______ of item name by item name
set item name to item name
sum of item name
sum of item name by item name
sum of expression
sum of expression by item name
```

SECTION 8
REPORT GENERATION

The Multics Simplified Filing Facility provides the capability for preparing reports from the data in your card file or subset. The program lets you define a report title and column headings, to specify the exact layout of columns, to sum some of the columns, and to break when certain items change, getting subtotals and grand totals.

In addition, you can use the "labels" command to produce mailing labels from the data in your card file.

THE PRINT COMMAND

The print command lets you prepare reports from your card file or subset in any of the formats described below.

A Simple Print With No Specifications

The simplest form of the print command is:

```
print
```

This causes the cards in the current subset to be printed out in the same format as the "show me " commmands. When you invoke this simple form the program asks you four questions:

```
HOW MANY CHARACTERS WIDE IS YOUR PAPER?
WHAT IS THE NAME OF THIS REPORT?
DO YOU WISH A SEPARATE PAGE FOR EACH CARD?
DO YOU WANT SINGLE, DOUBLE OR TRIPLE SPACING?
```

Usually you would answer 90 to the first question, but if you have been using a wide carriage terminal, or a narrow one, your answer could vary. It all depends on how much space you need to output your cards in the same form as they were input. The narrowest paper allowed when using this simple format is 40 characters.

You can give any name you choose as the name of the report. This is only needed to distinguish it from other reports you might be producing. Some response to that question is necessary, however.

As the report is produced on your terminal it is also being stored as a Multics segment. When the entire report is produced, you are asked if you want to have it routed to the printer. If you respond "yes", the segment is printed and deleted. Otherwise you are given the name of the segment and its eventual disposition is up to you.

Specifying An Item List

To print an item list, use this form of the print command:

```
print item name(s)
```

This causes the items in the list to be printed, one per line. It also invokes some of the other features of the print command such as headers and titles. You are asked the following questions:

```
HOW MANY CHARACTERS WIDE IS YOUR PAPER?
ENTER YOUR TITLE OF ONE TO THREE LINES.  END THE LAST WITH AN *.
DO YOU WANT A HEADER LINE WITH ITEM NAMES AS LABELS?
DO YOU WISH A NEW PAGE FOR EACH CARD?
THEN HOW MANY LINES DO YOU WANT PER PAGE, MAXIMUM?
WHAT IS THE NAME OF THIS REPORT?
DO YOU WISH TO PREVIEW THIS REPORT?
```

In addition, if one of your items was a multiple line, it would point this out and ask if you wanted it all printed.

If you provide a title, such as:

```
CURRENT FACULTY ROSTER
HOME ECONOMICS DEPARTMENT
MARCH 4, 1983
```

This is centered and printed at the top of each page. If you do not want a title, simply type an asterisk (*) and carriage return.

If you answer "yes" to the question about header lines, you are asked if you want the headers centered or left justified. The usual response is 'centered', but sometimes you might prefer to have the header start in the same column that the item does. If you answer "no" to this question, you are asked if you want to supply headers. Again you are asked if these headers should be centered. You are allowed up to three lines containing the headers. These lines are followed by a line of dashes.

If you answer "yes" you want a new page for each card, as you would do for address labels, you get the data for each card on a page by itself. If you answer "no", it asks you how many lines you want per page. For most output, this should be 60.

Printing Several Items on a Line

If you want to print several items on a line, specify two pieces of information: the column in which the item is to start printing and the number of columns that item requires. This is done using the following syntax:

```
print item name(col_start,col_width), item,...
```

For example:

```
print name(10,30), course(42,10), subject(55,30)
```

You are asked the same questions as in our previous example.

If you accidently try to overlap fields by specifying a starting column within the range of a previous field, the program changes the starting column of the field and give you a message that it is doing so. This adjustment can cause overlap and subsequent correction in following fields.

Multilined items are again noted, and you are asked if you want all lines printed. If you answer "yes", you are asked if they should be printed "as entered"; that is, each line on the card becomes one line in the output, or "to fit", where line separations are disregarded and as much of the text that fits in the space allotted is printed on each line. One of the quoted phrases above is the proper response. You can also cause separate lines, as they appear on the card that is, to appear in different columns on the report. (See "Formatting" below.)

Using Multiple Lines

In addition, you might want to print more items than can comfortably fit on one line or otherwise want to print different items on different lines to create a particular format. This you do by specifying a third parameter as follows:

```
print item name(line_no,col_start,col_width)
```

Note that no line number can be lower than any specified previously, but since item names can be specified in any order, and even duplicated, this should be no problem. Also, the separation between cards is determined by the line on which the first item starts. This is true even though the first item (or even the first line) might not print for every card since duplicate items at the beginning of the line(s) do not normally print. (See "Formatting" below.)

Note that if one number is specified, it is assumed to be the starting column; two are assumed to be the starting column and the number of columns. Only when three numbers are specified can you include the line number. If the line number is not specified, it is assumed not to have changed.

Specifying Control Items

On some reports, you might want a break in the listing when some items change. With the Multics Simplified Filing Facility, you can specify up to five items on which to break. The syntax is:

```
print item name(s) break on item name(s)
```

If you specify such breaks, you are asked if you want to skip to a new page when that item changes. This allows you to separate reports easily. For example:

```
print ...  break on course,subject

HOW MANY CHARACTERS WIDE IS YOUR PAPER?  65
ENTER YOUR TITLE, ONE TO THREE LINES.   END THE LAST WITH AN *
STUDENT LISTS
by
COURSE AND SUBJECT*
DO YOU WISH TO START A NEW PAGE WHEN THIS ITEM course CHANGES? yes
DO YOU WISH TO START A NEW PAGE WHEN THIS ITEM subject CHANGES? no
DO YOU WANT A HEADER LINE WITH ITEM NAMES AS LABELS? yes
HOW MANY LINES DO YOU WANT PER PAGE, MAXIMUM? 60
:
:
```

The items you are breaking on do not have to be items that you specified to be printed.

Summing Items

To specify that you want to sum certain items, type:

```
sum on item name(s)
```

at the end of your line (note that multiple item names must be separated by commas). For example:

```
print name(10,20), classroom hours(30,20), sum on classroom hours

or

print course(10,20), subject(30,20), no.  of students(50,20) break on course
sum on no.  of students
```

Note that the program does not let you sum a field that is not being printed.

If you have several breaks, the sums for each item are printed. For example, if you broke on course and subject and summed no. of students, you would get a total for each subject and a total for each course and an overall total.

FORMATTING

Generally, fields are printed as they appear on the cards. There are two exceptions: If a field is summed, it is converted to numeric form as specified; fields at the beginning of the line that are duplicated on the previous card are not normally printed. To assure that a field (and all after it) is printed, even if duplicated, place the characters [fp] (for force print) in the definition field; i.e., vendor[fp](30,12). Similarly, to specify that a particular item should be right justified in the space allotted to it, place the characters [rj] (right justify) in the field thus, vendor[rj](30,12). Both can be used in the same field, but only in that order.

If you require different lines in a multilined item to appear in specific places, such as in different columns, you can specify a particular line using the characters [ln], where n is the line number. For example, assume that you have name and address* as two items and you want the form:

```
     name     street     city and state
```

where street is usually the first line in address* and city and state the second. To get them to print on one line you would use:

```
     print name(10,30), address[l1](40,20), address[l2](60,20)
```

Using this convention, you can take any line from the item and place it anywhere in the report.

Finally, text can be included in your output by replacing an item name with the desired text enclosed in quotes. For example:

```
     ship name(10,20), capacity[rj](30,10), "short tons"(40,11)
```

OBTAINING A PRINTOUT

If you answer "yes" to the question "DO YOU WISH TO PREVIEW THIS REPORT?", the command print shows you on your terminal exactly what your output looks like, if possible, except that you see an X at the left of the page where a new page would start. At the same time, it is generating a Multics segment that can be used to get a printout. When it has completed, it asks you if you want a printout. If you answer "yes", the segment is sent to the printer and then deleted. If you answer "no", you are told the segment name, and you can do with it as you wish.

MAILING LABELS

You can use the card file "labels" command to produce mailing labels from the data in your card file. The syntax is:

```
     labels list
```

where list consists of the item names, separated by commas, in the order in which they are to appear. For example:

```
labels name,street,city
```

In this case each item will appear on a separate line. (Presumably city contains the city name, the state, and the zip code.)

If one of the items is a multilined item, such as address, each line in that item will be printed as it was entered. Thus,

```
labels name,address
```

would also produce proper labels if address contains the full address in a reasonable form. It should be noted that multilined items appearing anywhere in the list except at the end could cause some extra spaces in adjacent labels.

Finally, you might want to group items on one line. For example:

```
labels name,street,(city,state,zip)
```

causes the three items city, state, and zip to appear on the same line. They will be separated by a space and a comma.

The response to the labels command is the question, "HOW MANY LABELS ACROSS DOES YOUR FORM HAVE?". Your response depends on the form you are using. The program understands the responses 1, 3, and 4, and arranges output for these three standard forms. At the moment any other answer is rejected.

You are asked if you want your output routed to the printer. Normally your answer will be "no", because you have to arrange to have your output produced on the correct form.

NAME FORMS

When the labels command is used, the program assumes that the first item is the person's name. The program attempts to print the name in standard form: first name last name, suffix. On the card, the name could appear in either of the forms: first name last name, suffix or last name, first name, suffix. For example:

```
Jones, John, Jr
or
John Jones, Jr.
```

Other forms of a name result in readable but nonstandard forms.

COMMANDS INTRODUCED IN THIS SECTION

The following is a list of the print commands introduced in this section. Note that multiple item names must be separated by commas.

```
labels list
print
print item name(s)
print item name(s)(col_start,col_width)
print item_name(line_no,col_start,col_width)
print item_name(s) break on item name(s)
```

SECTION 9

CARD FILES FROM MULTICS SEGMENTS

The Multics Simplified Filing Facility lets you transform a Multics segment into a card file. This is useful if you have a segment that you want to put in card-file format for easy selection and retrieval of data, as well as any other useful file system function. To accomplish this, use the read command as described below.

THE READ COMMAND

If you have an existing file of data that you want to convert to a card file, you can do so by using the read command. The syntax is:

```
read segment name into file name
```

where *segment name* is the name of the Multics segment that currently contains the data, and *file name* is the name you give the card file.

The Filing Facility responds:

```
GIVE ME THE ITEM NAMES AND THE COLUMN LAYOUTS
```

Your response should be the list of items with each item followed by two numbers, in parentheses, which indicate the column that item starts in and the number of columns it uses.

For example, if you had a file that contained:

```
part number...     in column 1 through column 10
part name...       in column 11 through column 40
vendor number...   in column 41 through column 50
```

in a file called parts, and you wanted to make a card file from it, your command sequence would be:

```
read parts into parts

GIVE ME THE ITEM NAMES AND THE COLUMN LAYOUTS

part number(1,10), part name(11,30), vendor number(41,10)
```

Note that the read command cannot handle multiple line items. There must be no extraneous lines such as blank lines or title lines in the file. These lines are not detected by the read command, but cause bad entries in the card file.

Also, each line in the file must begin and end with a quote. This can most easily be done by the following sequence:

qedx	(invoke the qedx editor)
r file_name	(read in the file)
1,$s/^/"/	(put a quote in front of each line)
1,$s/$/"/	(put a quote at the end of each line)
w	(write the new file with old name) or
w new file_name	(write the new file with a new name)
q	(quit)

The file is now ready.

SECTION 10

MULTICS SIMPLIFIED FILING FACILITY PROGRAMS

After you have used the Multics Simplified Filing Facility for a time, you will find yourself writing down lists of instructions that you use to produce particular reports or to perform some other functions. These lists of instructions can be thought of as Multics Simplified Filing Facility programs.

Rather than having to find the right piece of paper each time you want to produce a particular report, it would be more convenient to store these lists as card files. It would be even more convenient if you could then "run" those sets of instructions "automatically". You can, using the run command.

THE RUN COMMAND

The run command lets you execute a card file containing a series of Filing Facility requests. When you create the card file to contain the card file requests to be executed, you must define one name on the card called "instruction"; optionally, you can include a "comment" field.

For example, to create a card file that could contain Filing Facility requests, you would enter the following Filing Facility commands after invoking the Filing Facility. Note that your input is preceded by an exclamation point (!) for clarity only and is not required as part of the input.

```
> !create a new file

Give me the name of the new file:  !card_command
How many items do you plan to have on each card:  !2
Give me the first name:  !instruction
give me the second name:  !comment*

> !save this file
>
```

Once created, you can then enter Filing Facility requests into the "card command" file.

For example, if you frequently create sorted lists of a personnel file, you would issue a series of Filing Facility requests similar to the following:

```
instruction: !open personnel file

comment*:    !This program sorts the personnel file by name
             and department and prints out department rosters.*

instruction: !sort on name

comment*:    !inner sort*

instruction: !sort on department

comment*:    !outer sort*

instruction: !print department(10,20), name(32,20)
```

The run command can execute all of the Filing Facility requests contained in the "card command" file. The syntax of the run command is:

run *file name*

After invoking the Filing Facility, the following instructions would execute the requests contained in the "card command" file:

open *card command*
run *card command*

Nested Run Commands

The file that contains the file system instructions could itself open and run another card file. This lets you break complex file system programs into smaller parts or, more importantly, have several programs share common instructions. Up to nine levels of nested runs are allowed.

Using Arguments in Multics Simplified Filing Facility Programs

You might find that you are producing several file system programs that have similar sequences of instructions but operate on different files or fields. In this case, you might prefer to have one set of instructions with dummy arguments to be filled in during execution of the program. The real arguments are transmitted via an argument list in the run command. The syntax is:

run *file name (argument1, argument2, etc.)*

Up to nine arguments can appear in the argument list. The first argument replaces all instances of the dummy argument &1 in the program file; the second all instances of &2, etc.

The following is an example:

```
> open new roster program

4 CARDS IN THIS FILE

> index of instruction

0001 open &1
0002 sort &2
0003 sort &3
0004 print &3(10,20),&2(32,30)

> run new roster program (personnel directory,name,department)
```

This produces the same results as in the example above. As the command in the first card is interpreted, the &1 is replaced by the first argument in the argument list, "personnel directory". The instances of &2 and &3 are replaced by "name" and "department", respectively.

Dummy arguments can appear anywhere in the command line. Even the command itself can be a dummy argument. The following are all legitimate, assuming the form of the run is correct:

```
sort on &1
&1   &2
run program_&1 (&2>&3, index of name)
&3
print &1(&2,&3),&4(&5,&6)
```

MULTIPLICATION AND EXPONENTIATION IN PROGRAMS

Because card files require an asterisk (*) to indicate the end of a multiline item, you must use the colon (:) as the multiplication symbol in all expressions. Similarly, two colons (::) must be used in place of the double asterisk (**) for exponentiation.

SECTION 11

CARD FILE COMMAND ABBREVIATIONS

The Multics Simplified Filing Facility lets you define your own set of abbreviations for long Filing Facility command lines. These abbreviations are stored in a card file called "abbreviations" which contains two item names called "abbrev" and "meaning".

DEFINING AND USING ABBREVIATIONS

To define Filing Facility abbreviations, you have to create a card file named "abbreviations" and define two item names for each card called "abbrev" and "meaning" as shown below. Note that the information you must supply is preceded by an exclamation point(!) for clarity only and is not required as part of the input.

```
> !create a new file

 Give me the name of this new file:   !abbreviations
 How many items do you plan to have on each card?:   !2
 Give me the first name:  !abbrev
 Give me the second name:  !meaning
```

Once you have created the file and defined the item names, you can begin entering your "abbrev" and "meaning" on each card. In the example below, an abbreviation "dis" is created for the command line "display the open file information"

```
> !enter a card

  abbrev:              !dis
  meaning:             !display the open file information
```

Do not forget to save the file once you have entered your data:

```
> save this file
```

To use an abbreviation, precede it with two periods(..). For example, to use the abbreviation created above type:

```
> ..dis
```

The Filing Facility responds with information about the current open file which, in this case, is information about the "abbreviations" file.

```
display the open file information
 There are  1 open files.  They are:
          abbreviations
 The active file is abbreviations

>
```

Abbreviations can only be used in command lines (though those command lines can be in Filing Facility programs) or in definitions of other abbreviations. Remember to precede an abbreviation by two periods thus "..dis". The two periods indicate that the word following is an abbreviation. That word is looked up in the abbreviation file, first opening that file if necessary, and its meaning is substituted in the command line for the abbreviation.

If the Multics Simplified Filing Facility encounters an undefined abbreviation, it asks you to define it then and there. It then uses this definition in reconstructing the command line.

Abbreviations can be of any reasonable length, but there can be no spaces in an abbreviation.

When Do You Use Abbreviations

You can use your own abbreviation for any Multics Simplified Filing Facility command that you use frequently and find bothersome to type. You can also use them for entire command lines that you use frequently; such as:

```
print part no. (10,10), part name(20,30), quantity(50,12)
```

which could be abbreviated as "print parts", for example. In the second case, the abbreviation not only saves keystrokes, but also makes it unnecessary to remember the formating information.

SECTION 12

ENCODING INFORMATION

Although Multics provides a secure operating environment, there can be occasions when you want to encode information to ensure confidentiality. This can be done using the "encode" command and a key of your own choosing. The syntax is:

```
encode item name key is key name
```

The encoding process uses an algorithm, a coding table and your key to change the character string to an encoded form. In this new form, the first character is always the vertical bar character "|". This character is used by the corollary command "decode" as a signal that the item really is encoded. If it does not appear, decode does not change the string. (This protects against your accidently using "decode" when you mean "encode".)

To decode the item, the syntax is:

```
decode item name key is key name
```

where the same key used to encode must be used to decode. This key should, therefore, be something you are going to remember, such as your favorite aunt's name. (it should not be the name of the item). It can be any length, and can even be missing, though it not recommended.

PART II

SECTION 13

INTRODUCTION TO THE MULTICS SPREADSHEET PROGRAM

The Multics Spreadsheet Program (MSP) is a computer program that lets you devise solutions for that large class of problems that can normally be laid out on a sheet of paper providing rows and columns. MSP provides you with an electronic sheet via your computer terminal. The electronic sheet has 26 horizontal rows (labeled a through z) and 99 vertical columns (labeled 1 through 99). Your sheet can alternately be 99 rows by 26 columns, as explained later.

The MSP row/column combination provides 2,574 cells (or boxes) each of which you can refer to (or address) by its row and column position. For example, the first cell in the first row is called a1; the last cell in the last row, z99; the third cell in the third row, c3; and so on.

CELL CONTENTS

Originally every cell is undefined and has a value of zero. You can enter into a cell in the sheet one of the following:

- a number
- an expression
- a comment

As you build up your spreadsheet by filling in cells with comments and expressions, you can set up a complex trail of references where one cell's contents depend on others which depend on others and so on. Each time the sheet is displayed, each cell's contents is computed in turn. Thus, changing the contents of one cell causes all the other cells whose value is related to those contents to also be changed.

It is this dynamic nature that gives MSP its value. It allows you to set up a general model of a situation, your budget perhaps, and then "plug in" the actual values for a specific situation. Or, it lets you develop a model and then ask "what if" questions, such as what if interest rates go up 2%?

Entering Numbers

Entering a number such as 123.45, 0.0078, etc. in response to the INPUT prompt causes that number to become the value associated with the cell you specify. The number, or a rounded approximation to it, appears when the sheet is displayed.

MSP lets you enter numbers in several different forms, depending on which is most natural to you. For example, 1000000; 1000000.00; 1,000,000; 1 000 000; all result in one million being entered. You can enter the dollar sign ($) but it is ignored. Additionally, MSP lets you use the percent (%) sign. Both 10% and 0.10 result in 0.10.

Some forms that appear reasonable, however, can surprise you. For example, -3 is not a number; it is an expression because it contains the operator "-", the minus sign.

THE ARITHMETIC OPERATORS

There are five arithmetic operators recognized by MSP. They are plus (+) for addition, minus (-) for subtraction, times (*) for multiplication, divided by (/) for division, and raised to the power (^) for exponentiation. MSP does not allow two operators to be adjacent in an expression. None of these operators has precedence over the others, expressions are simply evaluated from left to right. Because of this it is sometimes necessary to group (or partition) some of the terms in your expression using parentheses. For example:

```
2 * 3 + 4  =  10

2 * (3 + 4)  =  14

10 ^ - 2     is illegal (two operators in sucession)

10 ^ (- 2)   =  0.01
```

THE CONDITIONAL OPERATORS

There are three conditional operators, equal to (=), less than or equal to (<), and greater than (>). These operators provide only a TRUE or FALSE result and are used in conditional expressions which are explained in a later section.

= (equals)
: a1=a2 results in 1 (TRUE) if a1 equals a2. Otherwise the result is 0 (FALSE).

< (less than or equal to)
: to a1<a2 results in 1 (TRUE) if a1 is less than or equal to a2. Otherwise the result is 0 (FALSE).

> (greater than)
: a1>a2 results in 1 (TRUE) if a1 is greater than a2. Otherwise the result is 0 (FALSE).

Although single variables are given in the examples above the arguments can be expressions. As with all expressions, they are evaluated from left to right. For example:

```
2*a1 > (a3/4) * a5
```

results in a 1 (TRUE) only if two times the value of a1 is greater than the value of a3 divided by 4 otherwise the result is 0 (FALSE). Consequently the product of this result and a5 is the value of a5 (1*a5) or zero (0*a5).

Expressions

An expression can consist of:

1. a variable (cell name) by itself. For example, input = a3 constitutes entering an expression.
2. numbers joined by arithmetic operators, for example, 2 * 3 + 4
3. a single number preceded by a sign, for example, -3.
4. a function or a function preceded by an operator, for example, &sin (3.147) or - &sin (3.147).
5. any combination of variables, numbers, and/or functions joined by the arithmetic operators and, perhaps, grouped by the use of parentheses, for example, 3 * (a1 + a2) + &min(a1;a2).

Entering an expression such as 2+3.46, 22/7, 1+2+3+4+5, etc., associates that expression with the specified cell and that cell's value becomes the computed value of the expression.

However, expressions can also refer to other cells such as 2*a1, a2+a3, a7/(a1+a2), etc. In which case, the values used are the values of the specified cells whenever the expression is evaluated.

Conditional expressions allow you to alter the value set or the expression used according to some condition. The form is:

```
condition {TRUE Clause} {False Clause}
```

where the condition equates to either 1 (TRUE) or 0 (FALSE) using the conditional operators.

Comments

Comments are entered by preceding them by an apostrophe ('). For example:

```
INPUT = 'This is a comment.
INPUT = '  Income
INPUT = '  Gross Profit Before Taxes
```

Comments can be up to 30 characters long, but usually only the first part of the comment appears when the sheet is displayed. For appearances, you may want to include one or two spaces as the first characters in a comment.

SECTION 14

USING THE MULTICS SPREADSHEET PROGRAM

The Multics Spreadsheet Program (MSP) runs on the Multics system. To use it, you must be a registered user on that system.

LINKING TO MSP

Once you have become a Multics user and have learned how to get on the system, you might have to make a "link" to MSP. This depends on where MSP is located on your system. Try entering MSP, as described below, prior to entering the link command. If you are successful, you can skip the linking step.

To link to MSP, enter the following Multics command:

```
link path
```

where path is the pathname of the location of MSP on your system.

INVOKING MSP

Start MSP by simply typing:

```
msp
```

The msp command invokes the Multics Spreadsheet Program and begins the series of prompts described below.

USING MSP

Once invoked, MSP responds by displaying:

```
CELL
```

MSP waits for you to type in a cell name such as a1, a10, b67, etc. (no blanks are allowed in the cell name). This becomes the "current cell". After you enter a cell name and a carriage return, MSP displays:

```
INPUT =
```

At this point, you type in the number (constant), comment, or expression that you wish to put in that cell and follow this by a carriage return. Depending on the version you are using and what you've typed, MSP recomputes all of the values in the sheet and prints some portion of it on your screen.

Once you have entered a value for input and entered a carriage return, MSP once again prompts for a cell name:

```
CELL
```

MSP continues to prompt you for cell name and input letting you enter values and get new results.

EXAMINING INDIVIDUAL CELLS

When the spreadsheet is displayed only the values or comments appear and there is no indication of whether a value was entered as a constant or is the result of evaluating the expression in that cell. The simplest way of examining an individual cell is to simply name it in response to MSP's cue of:

```
CELL
```

If that cell has been defined its contents is printed before you receive the INPUT= cue. For example, assuming that you have defined cell a9 to contain the expression 3*(a1+a2) and that the current evaluation of that expression is -76.89, if you enter the cell name a9 in response to the CELL prompt as shown below the following is displayed:

```
CELL a9
currently contains 3.*(a1+a2) = -76.88999999
INPUT =
```

If the cell contained a comment, the entire comment, all 30 characters, are printed. The other way to examine a cell is using the $debug command, but that is described in a later section.

MORTGAGE EXAMPLE

The following example illustrates the procedures used for creating a mortgage analysis spreadsheet. There are references to topics not covered yet, such as replication (%r), and absolute and relative references which are explained in the next section. The example includes comments in the righthand column, which are in uppercase, to explain the procedures being used and are not required as part of the input. The input you supply is preceded by an exclamation point (!) for clarity only and is not required as part of the input.

```
msp                                      INVOKE MSP

CELL !a1                                 START WITH CELL A1 AND
INPUT = !45,000                          ENTER THE PRINCIPAL

CELL !a2                                 ENTER THE CURRENT VALUE OF
INPUT = !84,000                          OF THE HOUSE

CELL !b1                                 ENTER THE INTEREST RATE
INPUT = !13%

CELL !b2                                 ENTER THE NUMBER OF YEARS
INPUT = !30
```

So far, the spreadsheet has not been shown, but it is displayed each time you enter data in response to the INPUT prompt.

```
CELL !b3                                 THIS EXPRESSION COMPUTES
INPUT = !b1*a1/(1-((1+b1)^(-b2)))        THE ANNUAL PAYMENT


CELL !c1                                 THIS EXPRESSION COMPUTES
INPUT = !1+b1*a1-b3                      THE REMAINING PRINCIPAL


CELL !c2                                 THIS EXPRESSION COMPUTES
INPUT = !(1+b1)*c1-b3                    THE REMAINING PRINCIPAL
                                         AFTER THE SECOND YEAR

CELL !c2                                 NOTE THAT CELL C2 HAS BEEN
currently contains                       DEFINED THUS ITS CONTENTS
(1.00+b1)*c1-b3    44640.435059          ARE DISPLAYED

INPUT = !%r                              REPLICATE C2 TO PRODUCE 28
replicate from where to where?           REPLICATION OF THIS EQUATION
give me the cells. !c3 !c30

(1.00+b1  !a                             DECLARE B1 AN ABSOLUTE REFERENCE

(1.00+b1)*c1 !r                          DECLARE C1 A RELATIVE REFERENCE

(1.00+b1)*c1-b3 !a                       DECLARE B3 AN ABSOLUTE REFERENCE

CELL !d1                                 THIS EXPRESSION COMPUTES THE
INPUT = a2-c1                            EQUITY AFTER ONE YEAR

CELL !d1                                 DISPLAY THE CONTENTS OF CELL D1
a2-c1         39169.207031

INPUT = !%r                              ISSUE THE REPLICATE COMMAND TO
replicate from where to where?           REPLICATE D1 IN CELLS D2 TO D30
give me the cells. !d2 !d30
```

```
a2 !a                                       DECLARE CELL A2 TO BE AN ABSOLUTE
                                            REFERENCE

a2-c1 !r                                    DECLARE CELL C1 TO BE A RELATIVE
                                            REFERENCE
```

Your spreadsheet should now look like this:

```
          1           2           3           4
  a   45000.00    84000.00
  b       0.13       30.00     5794.21
  c   44830.79    44640.44    44426.28    44185.36
  d   39169.21    39359.56    39573.72    39814.64
  e
  .
  .
  .
  m
```

Once you have finished entering the data, check to see that the replications performed on cells c3 and d3 are correct as follows:

```
CELL !c3

(1.00+b1)*c2-b3         44426.282227
INPUT =

CELL !d3
a2-c3        39573.717773
INPUT =

CELL !$save

Give me a file name to use.!mortgage

mortgage saved

CELL !$quit
```

At this point you would be returned to command level within Multics.

SECTION 15

MSP FUNCTIONS, COMMANDS, AND CONTROLS

MSP provides you with an extensive library of functions, commands, and controls. The majority of these are described in this section; additional financial functions, graphics plotting commands, and debugging commands are described in subsequent sections.

THE FUNCTIONS

You can use MSP functions in expressions exactly like you would use any constant or variable. MSP functions are either single argument functions like sine and cosine, or functions that work with a list like the sum function. The functions are:

&sin(...)	computes the sine of the argument, such as &sin(5.14167/8) = 0.38
&cos(...)	computes the cosine of the argument
&sqrt(...)	computes the square root of the argument if it is positive, otherwise this function returns zero.
&exp(...)	computes the exponential of the argument if the argument is less than 88.00, otherwise it returns 100,000,000.00.
&ln(...)	computes the natural log of the argument. If the argument is negative it returns the log of the absolute value of the argument (in preference to zero); if zero, it returns zero.
&sum(list)	computes the sum of the values contained in the list.
&max(list)	finds the maximum of the defined values in the list.
&min(list)	finds the minimum of the defined values in the list.
&mean(list)	computes the arithmetic mean of the values contained in the list. &mean is recognizes undefined cells and ignores them. Note that functions and expressions in the list are considered to contribute one item each.
&count(list)	returns the count of defined items in the list.
&choose(n;list)	returns the nth item in the list where n can be a constant, a variable, or an expression. (If n is an expression it must be enclosed in parentheses.) If n is undefined, zero, or negative or if n is greater than the length of the remaining list, zero is returned.

&match(value;list)	matches the value given against the list and returns the position in the list that matches that value, if any. If no match is found, a zero is returned. If value is an expression, it must be enclosed in parentheses.
&match_nth(n;value;list)	matches the value given against the list and returns the position in the list at which the nth occurance of that value appears, if any. If there are not at least n occurances of the value in the list, zero is returned. If n or the value is given as an expression, it must be enclosed in parentheses.
&lookup(value;range1;range2)	looks up the value in the first table of values specified by range1 and selects a corresponding value from the table of values specified by range2. If the value given lies between two values in the first table, the value returned lies an equal distance between the two values in the second table. (This is sometimes called interpolation.) The values in the first table must be arranged in ascending order! If value is an expression, it must be enclosed in parentheses.
&integer(...)	returns the integer portion of the argument. (&integer(3.2*2) returns 6; &integer(-3.2*2) returns -6.)
&even(...)	returns a one (TRUE) if the integer of the argument is even, a zero (FALSE) if it is odd.
&abs(...)	returns the absolute value of the argument.
&pi()	returns 3.141596 (an approximation to the value of Pi). Note that the parentheses are needed even though no argument is used.

Functions can appear anywhere in an expression. For example:

```
&exp(3*&ln(&sqrt(a1+a2)))
```

The forgiving feature of MSP is that it lets you be careless about closing the parentheses at the end of an expression.

```
&exp(3*&ln(&sqrt(a1+a2
```

would work as well. You can also include spaces. They are removed before the line is analyzed.

Function Ranges

Some of the MSP functions, such as &sum (...), use ranges or lists for their arguments. A range specifies the cells which are to be included in the summation and is expressed by cell1:celln. cell1 is the first cell in the block of cells (the upper left hand corner), celln is the last cell in the block (the lower right hand corner), and the colon indicates that this is a range.

For example, a3:c6 includes the cells a3, a4, a5, a6, b3, b4, b5, b6, c3, c4, c5, and c6.

Function Lists

Lists are made up of a series of ranges, numbers, cells, functions, and/or expressions with items in the list separated by semicolons. Some examples:

```
&sum(a1; a10; b3:b7; 43; (3 * a5))
&mean(1; 2; 3; 4; b3:b7)
&min(a1:a10; &sum(b1:b6))
```

Note that expressions, such as 3 * a5 in the first example, must be enclosed in parentheses when appearing in lists. Note also that the smallest list can be a single range. A single variable, constant, or expression generally does not constitute a list (following a single item by a semicolon does make it a list).

Lists can only be used as the arguments of certain of the functions. By itself a list has a value of zero. If you get this result after entering what you consider a legitimate function expression, you should examine your expression carefully. For example:

```
&sum(a1:a5); &mean(b12:b17); a14)
```

is a list of three items, &sum(a1:a5), &mean(b12:b17), and a14 and its value is zero.

```
&sum(a1:a5; &mean(b12:b17); a14)
```

provides the sum of the contents of cells a1 through a5 plus the mean of the contents of the cells b12 through b17 plus the contents of a14.

THE COMMANDS

You can enter a command instead of a cell name when MSP asks for the cell. A command produces a result that is reflected globally and does not use any particular cell as a parameter. The commands are:

$quit	terminates the MSP session.
$write	save the contents of the spreadsheet as it currently exists. $write asks you for a name (twenty or fewer characters) and writes the necessary information into a Multics segment called: name.msp.saved in your directory.
$save	similar to $write except that if the name of the sheet is already known it does not ask you for a name. $save can be used exclusively to save the sheet. $write is

	provided only to allow you to save a sheet under a different name than the one currently known. The name of a sheet, if it is known, is printed out above the cue "CELL".
$load	is the inverse of $save and lets you retrieve spreadsheets that were previously $saved. If you give $load a name which does not match any of the saved sheets, MSP lists all of the segments in your directory that are named name.msp.saved and you can then select the correct name.
$fresh	gives you a fresh (empty) sheet.
$erase	erases (deletes) the saved version of this sheet. You should normally use $erase if you no longer need that sheet. It is not necessary to do an $erase before a $save or $write.
$index	gives you a list of the Multics segments in your directory that contain saved sheets. These segments have the name [sheet_name].msp.saved.
$eval	allows the immediate evaluation of an expression that may or may not contain references to cells in the sheet.
$calc	changes the order of calculation from row by row to column by column. Row by row is the default. (See Order of Calculation below.)
$cw	changes the width of the columns as displayed on the screen. The MSP default is 12 characters. $cw lets you increase this to make comments more readable or decrease it to allow more columns to appear on the screen. Normally, decreasing the column width to less than 9 is not recommended since 9 spaces are required to display numbers in scientific notation when they do not fit in the space alloted.
$noc	specifies the number of columns that you want displayed. The MSP default is five columns, but if you have a terminal which has more than 80 character lines or have changed the column width you may want to increase this.
$np	specifies the number of decimal places you want displayed.
$round	specifies that results should be rounded to a specific number of places. MSP attempts to truncate fractions after rounding. Rounding is useful when duplicating systems where such rounding is naturally done; for example, when using dollars and cents.
$paper	changes the form of your sheet to 99 rows by 26 columns (long) or 26 rows by 99 columns (wide). Wide is the default.

$print	copies the sheet in a form suitable for printing into a segment named name.msp.listing. As in $write, you are asked to supply the name. You are also asked if the segment should be directed to the printer. The form of the output is determined by the parameters set by $noc, $cw, and $np at the time the $print command is invoked. In particular, the number of columns per printed page is determined by $noc. As a security device, any row or col which has a colon (:) in the first position of its label does not have the contents of its cells printed (see %label below).
$list	compiles a complete listing of each cell's contents - constant, expression, or comment - and places it in a segment called name.msp.list, where you are asked to supply name. You are asked if the segment should be directed to the printer. The list can be useful in checking out a model or simply for keeping a record of how it was put together.
$plot	generates plots from your spreadsheet. (See Section 18.)
$db	a debugging tool (see Section 19).
$memo	invokes the memo and report writing facility. (See Section 20.)
$clean	removes unused expressions from storage to allow the space to be reused. A cleanup is invoked whenever a sheet is $saved, but it might be necessary to use $clean if your model is a large one and you are still "bread-boarding it." Should be used if message MAXIMUM EXPRESSION SPACE EXCEEDED appears.

If an error message causes your terminal to become disabled (or "hung"), hit the BREAK key to return to Multics command level. Then type "pi" to return to MSP.

Except for the $quit command, all of these commands return you to the point where MSP prints out the CELL prompt.

MSP CONTROLS

You can enter any of the controls listed below when MSP asks for input. They use the current cell, the one that you just named, as either an index or an argument. They are:

%c corner	sets the current cell as the upper left-hand corner of the displayed sheet. The default corner is a1. However, if the current cell is outside the current frame, the frame shifts to include it.

%d duplicate	copies the contents of the current cell, constant or expression, into another cell. This differs from giving the name of a cell as input, which only copies the value, and from replicate (see below) which allows you to modify the expression.
%db duplicate block	copies a block of cells into a new location. Unlike %d, any expressions are modified to retain relative references within the block. It is recommended that this only be used to duplicate blocks of data or text.
%e enter	enters the information (numbers, comments, or expressions) into a block of cells in one input operation. MSP uses the current cell as the first in the block and asks you for the ending cell in the block. It then cues you by typing out the name of each cell for which the data is to be supplied. Hit the carriage return after each entry.
%ea enter/add	same as enter except that any numbers entered are added to the value currently in the cell. The cell becomes a constant regardless of its previous status.
%hcs hold columns	ensures that specified columns always appear when the sheet is displayed. You can, for example, specify a column containing the totals to print each time the sheet is displayed to see the effect of changes you make in the spreadsheet. MSP asks you how many columns you want to hold. To reset use %hcs and specify 0 (zero).
%hrs hold rows	the same as %hcs except that it lets you specify rows that are always displayed rather than columns.
%hb hold block	creates another "window" that shows the portion of the sheet specified as follows: The current cell is the upper left hand corner. The number of columns displayed is as currently specified by default of use of the $noc command. The number of rows are defined by the response to the question "How many rows?". This "window" includes both column and row labels and is not affected by any %hrs or %hcs that have been invoked. Held rows and columns still appear in the current window.
%ic insert column	inserts a blank column at the column location indicated by the cell name. MSP changes defined expressions so that they continue to refer to the same cell even though that cell, or the one containing the expression, has been

	moved. %ic also moves column Labels. (See %rc remove column.)
%ir insert row	inserts a blank row at the row location indicated by the cell name. As with %ic MSP modifies expressions and move labels. (See %rr remove row.)
%iter iterate	iterates the matrix calculations the specified number of times. You are asked for the numbers of iterations. (Used primarily for running simulations.)
%lr label row	assigns a label to a row
%lc label col	assigns a label to a column.
%lrs or %lcs	labels several rows or columns at once.
%mb move block	moves a block of cells around on the sheet. The cell name defines the upper left hand corner of the block to be moved. You are asked for the lower right-hand corner of the block and the upper left-hand corner of its new position. MSP adjusts all of the defined expressions so that the defined relations continue to exist. Both %ir and %ic are special cases of move block. (A mistake in using %mb can be disastrous. It is advisable to $save your sheet first, just in case.)
%r replicate	replicates the contents of the current cell through a specified block of cells. If the contents of the current cell is a constant, no further response is needed. If the contents of the current cell is an expression containing variables, however, MSP must know if those variables are absolute (a) or relative (r). Absolute means that the exact cell name is used in each replication of the expression; relative means that the cell name to be used is that which is in the same relative position as this variable is to the current cell.

In the mortgage example, we had to replicate the expression in cell c2, which showed the principal remaining after the second year, through cells c3 to c30. The expression replicates as follows:

```
c2 (1+b1)*c1-b3
c3 (1+b1)*c2-b3
c4 (1+b1)*c3-b3
       .
       .
       .
c30 (1+b1)*c29-b3
```

As you can see, certain cell names in the expression are "absolute addresses"; that is they remain the same for every replication. In cell c2's expression, however, the cell name c1 is "relative"; it refers to the cell "before this one" and this relation is preserved throughout the replication. This is a simple relation, "the cell before", but the relation need not be that simple. It could be "the cell three columns before and two rows above" for example. Once the initial relation is established, by your entering the expression in a specific cell, %r can preserve it.

To determine which variables are absolute and which relative, MSP processes and prints the expression stopping each time it hits a variable. It then waits for you to respond with either an "a" for absolute or an "r" for relative and a carriage return.

Additionally, in some cases you might want to hold only the row or column of the variable name constant (absolute). In these cases "ar" or "ra" would be the proper response.

%rc remove column	deletes the column to which you are currently pointing. The columns to the right move over to fill in the space. (Expressions referring to this column might be incorrect after a %rc operation.)
%rr remove row	deletes the row currently pointed at. The rows below move up to fill in the space (see warning under %rc).

%s
substitute

modifies expressions by substituting all occurences of one string of characters with another string. MSP asks you for the two strings and then lets you examine the result. Example:

```
CELL d1
currently contains  a1*b1+c2*(c2+a3)/c1
INPUT = %s
```

Enter the two strings

```
c2 c3
a1*b1+c3*(c3+a3)/c1
```

Does that look like what you want?

%tr
transfer

allows the values in a block in a saved spreadsheet to be copied into (to replace) or added to the designated position in the active sheet. The contents of any cell replaced or added to by this control is redefined as a constant regardless of its previous contents. (If it were a comment or undefined its value was zero.) This control takes a snapshot of the saved sheet at the moment it is invoked. There is no dynamic link established between the sheets so that future changes in one will not be reflected in the other.

```
CELL f5
INPUT = %transfer
```

Give me the name of the sheet you want to transfer from. sheet_name

Give me the first and last cells of the block to be copied. h20 j30

This would copy the numeric values (only) in the 33 cells in the block h20 to j30 in the saved sheet sheet_name to the corresponding block starting in f5 in the active sheet.

%w

Erases out a cell and sets it to undefined.

%wb

erases a block of cells.

The Conditional Expression

Occasionally it is useful to allow different values to be assigned to a cell or to have a different expression evaluated depending on some condition. One might say, "If the total sale is greater than $100 then the discount is five percent of the total sale; otherwise the discount is three percent of the total sale." In MSP the condition must be expressed as either true (expressed by a 1) or false (expressed by a 0). A simple condition statement might be, for example, just a cell name. If that cell contained a zero (or actually anything other than a one) the condition is considered false; if it contained a one it would be considered to be true. A complex condition would include one or more of the conditional operators <, >, and =. For example: a1 > b3 is true only if the value contained in a1 is greater than the value contained in b3. There is also one conditional function, &even, which can be used. &even(2) is true, &even(3) is false.

The conditional expression consists of a condition plus an expression to be evaluated if the condition is true (the TRUE expression) and, optionally, an expression to be evaluated if the condition is false (the FALSE expression). The form is:

```
condition {TRUE expression} {FALSE expression}
```

For example:

```
a1/3=94{a1+a2}{a1+a3}*(a1/2)
```

which says that if the result of the contents of a1 divided by 3 is equal to 94 then multiply (a1+a2) by (a1/2) otherwise multiply (a1+a3) by (a1/2).

```
a3*(&even(a2){a2+1}{a2})
```

which says multiply a3 by (a2+1) if a2 is even or by a2 if a2 is odd.

The true and false expressions can be nested and can themselves contain conditional expressions. Thus, the following could serve as a table look-up expression.

```
d1=a1{b1}{d1=a2{b2}{d1=a3{b3}{9999}}}

Which says:       if d1 = a1 then use b1
             else if d1 = a2 then use b2
             else if d1 = a3 then use b3
             else                 use 9999
```

In this case, if a match for d1 is not found in the table the value is set to 9999. Otherwise it is set to the corresponding value from row b. Note: as with parentheses you do not have to worry about trailing braces (}) if they appear at the end of the expression.

Note that if just a conditional expression is given with no following TRUE expression, the result is a 1 for a TRUE condition, or 0 for a false condition.

The Conditional Operators

There are only three conditional operators, the equal sign (=), the greater than sign (>), and the less than or equal to sign (<). Consequently, one has to use some imagination to simulate the and, or...operators. The following are suggested:

```
and:  (a1<b1)&(b1>c1) use a1<b1{b1>c1{exp}}
a TRUE within a TRUE

or:   (a1<b1)|(b1>c1) use a1<b1{exp}{b1>c1{exp}}}
a TRUE within a FALSE
```

The exclusive or is left as an exercise for the reader.

ERROR HANDLING

MSP detects errors, informs you of the error, and in some cases lets you make a correction. Occasionally something severe, such as accidently hitting the break key, can return you to the Multics command level. In that case, typing "pi" followed by a carriage return returns you to MSP at approximately the point you left it.

Numbers too large to be printed in the column width allowed are printed in scientific notation of the form:

```
d.dd EXX
```

where d.dd are the first three digits of the number and XX indicates how many places to the right the decimal point should be moved.

Order Of Calculation

There are two instances when the order of calculation is important. The first is within the expression; the second, the order in which the cells are evaluated.

Within the expressions the calculations proceed from left to right with no order of precedence among the arithmetic operators or functions. This means that 2+4*3 produces 18, not 14 as a FORTRAN programmer might expect. Parentheses can be used to properly group expressions and in effect set an order of precedence. For example, 2+(4*3) = 14. So does 2+4*3 by the way.

Within the spreadsheet (or matrix, if you prefer to think of it that way) the cells are normally evaluated from left to right, top to bottom (row by row from left to right). This could lead to some unexpected results if the value of a cell depends on the value of a cell that is evaluated later.

Occasionally you might prefer to think of your model as columnar and want the calculations to proceed from top to bottom, column by column. The command $calc lets you do that.

In some cases, it might be obvious that you prefer to have a long sheet (99 rows by 26 columns) and to have the calculations proceed column by column. In those cases you may prefer to use the entry point msp$c when invoking MSP.

The Internal Storage of Expressions

The expressions you develop while using MSP are stored in an array called the expression_array. Each cell that has an expression defined for it has a pointer to that expression in the expression_array. When an expression is replicated it is actually the pointer that is replicated and the expression as defined in the array is shared by the several cells.

Variables (cell names) in the expressions are originally defined by their relative position to the home cell. Thus, if in cell a1 you define an expression which uses the varibles a2 and b2, a2 are stored as 0,1 (which means plus zero rows and plus one column) and b2 as 1,1 (plus one row and plus one column). If while replicating an expression you define a variable as absolute, that variable is henceforth expressed by its absolute address. Otherwise, the relative position is maintained and the actual cell used changes as the home cell changes.

If you replicate a previously replicated cell, you should be aware of the expression_array and the shared pointers. Three things are important. One, you cannot change a variable back from absolute to relative. Two, changing a variable from relative to absolute changes it for all the cells that point to that expression. Three, replacing the expression in a replicated cell, even in the original cell, only changes the expression for that cell.

SECTION 16

DEFINING AN MSP MODEL

For casual uses of MSP, you can sit down at your terminal and develop your spreadsheet "on the fly". For any serious use of MSP, such as developing a model of the United States economy, for example, a little premeditation and planning can save a lot of aggravation and some serious mistakes. The following is a list of suggested steps you should consider before entering data into MSP.

1. First, list all of the elements that are going into your model.

2. Indicate whether each item is a constant; that is, has no dependency on any of the other items in the list, or a variable.

3. For each constant, write down an initial value and remember that you are able to change this value once your spreadsheet is set up. (The advantage of this step is that it helps you to get your units defined.)

4. For each variable write down the expression using the name that you assigned the elements in your original list. For example:

   ```
   sales_income = cash_sales + bank_card_sales +
   credit_sales - refunds
   ```

   (Now you can always go back and see what you meant.)

5. Quickly look over your list. Have you defined as constant something that should be variable or vice versa? Have you left anything out such as the totals that you want to develop?

6. On a large sheet of paper (such as accountants are fond of using), start laying out your sheet. Try to label rows and/or columns to identify some of the items in your list. Start entering your constants using the first row(s) and starting at the left of your sheet. At this point, you should write in the name of the constant rather than the value.

7. Start assigning your variables to cells. First, enter all of those that use only constants in their expressions (or cells that contain constants). As you assign them, rewrite your expression using the cell names that are now assigned to those constants. For example:

   ```
   sales income

   a3+a4+a5-b2
   ```

8. Assign the other variables by writing them in lightly; try to use only cell names in the expressions. If you assign variables from left to right and top to bottom and are always able to fill in the expression completely, you can skip the next step. If you found that you had to skip ahead and assign variables before you could complete any of your expressions, go to step 9.

9. Go through your variable expressions checking to see if any expression refers to cells that are to the right or below the cell in which the expression appears. If there are cells to the right or below the cell in which the expression appears, this is an inconsistency because of the way that MSP processes the sheet when recalculating it. Try to rearrange things until you have removed any such inconsistencies. If you cannot, you have a circular dependency situation.

10. Go to your terminal, invoke MSP, enter your model, $save your initial version, and try it out.

SECTION 17

FINANCIAL COMPUTATION

MSP provides eight financial functions that allow financial computations involving compound interest. The functions and their use are described in this section.

FINANCIAL FUNCTIONS

Each of the eight financial functions described below use as their argument list any three of the following four items:

```
A, amount
The dollar amount, such as the amount of a loan

R, rate
The interest rate per period

N, periods
The number of periods, months, years, etc., involved

P, payment
The amount of the payment per period
```

These items can be variables (cell names), constants, or expressions. If you use them as expressions, enclose them in parentheses. Note that the interest rate is expressed as "interest per period". If the number of periods is expressed in months and the annual interest rate is 15%, then rate can be expressed as 0.0125, 1.25%, (.15/12), or (15%/12).

Future Value

The future value returns the future value of amount A compounded at rate R over N periods. The equation used is: future value = A*((1.0+R)^N). The function format is:

```
&f_value(A;R;N)
```

Present Value

The present value returns the present or discounted value of a future amount A to be received after N periods when the interest rate is R. The equation used is: present value = A/((1.0+R)^N). The function format is:

```
&p_value(A;R;N)
```

Amount of Payment

The amount of payment returns the amount of the periodic payment necessary to repay a loan of amount A over N periods at an interest rate or R. (Also called capital recovery.) Can also be used to determine the annuity that can be paid from a given fund. The equation is: payment = (A*R)/((1.0-(1.0+R)^(-N)). The function format is:

```
&payments(A;R;N)
```

Present Value of an Annuity

The present value of an annuity returns the amount that would have to be deposited in a fully paid up annuity fund if that fund is to pay out the amount P for N periods and the interest rate is R. The equation used is pv_annuity = P*(1.0-((1.0+R)^(-N)))/R. The function format is:

```
&pv_annuity(P;R;N)
```

Future Value of an Annuity

The future value of an annuity returns the amount that would be in an annuity fund after N periods if amount P is deposited each period at interest rate R. The equation used is: fv_annuity = (P*((1.0+R)^N)-1.0)/R. The function format is:

```
&fv_annuity(P;R;N)
```

Rate of Interest

The rate of interest returns the rate of interest of a loan of amount A if N payments of amount P will repay that loan. This is also called the Discounted Cash Flow Rate of Return. This rate is computed by iterating the equation R(i+1) = P/A*(1.0-((1.0+R(i))^(-N))) until R(i+1) and R(i) are essentially equal. The function format is:

```
&rate(A;P;N)
```

Sinking Fund

The sinking fund returns the amount that must be placed in a fund at the end of each period at interest rate R to accumulate the amount A in N periods. Also called a Fixed Investment or Annual Installment. The equation is: payment = A*R/((1.0+R)^N-1.0). The function format is:

```
&sinking(A;R;N)
```

Number of Periods

The number of periods returns the number of periods required to repay a loan A at interest rate R with periodic payments of amount P. The equation is: periods = -1.0*(&ln(&abs(A*R/P-1.0))/(&ln(1.0+R)). The function format is:

```
&periods(A;R;P)
```

Note that The Discounted Cash Flow calculation used in buy vs. rent decisions is the present value of the purchase amount minus the present value of the annuity amount represented by the payments and anticipated interest rate. If the result is negative then purchasing is the correct decision.

SECTION 18
GRAPHIC PLOTTING

THE $PLOT COMMAND IN MSP

The $plot command lets you output in graphical form arrays that you have generated in MSP. As an example, suppose that you have generated a mortgage situation, have determined that the payment on your mortgage was x dollars, and have computed what the principal remaining would be for each year over y years. Your spreadsheet might look like this:

```
                      1          2          3          4          5
mortgage   a   45000.00
terms      b       0.12      30.00
payment    c    5586.46
periods    d       1.00       2.00       3.00       4.00       5.00
principal  e   45000.00   44813.43   44604.69   44370.79   441108.82

where:
a1 contains the initial mortgage.
b1 contains the mortgage interest rate.
b2 contains the number of years.
c1 contains the payment expression b1*a1/(1-(1+b1)^(-b2))).
The row d1 through d30 contains the period identifier.
The row e1 through e30 contains the principal at the beginning
of each term and is computed by replicating the expression (1+b1)*e1-c1.
```

To plot the principal vs the period, the following interactive plotting session would be used. Note that your input is preceded by an exclamation point (!) for clarity only and is not required as part of the input.

```
CELL  ! $plot
Do you want wide or narrow plots?  !narrow
```

Wide plots should only be generated on terminals having 132 characters per line.

```
Give me the name of the cell containing the first item in the x variable
!d1

Give me the name of the cell containing the last item in the x variable
!d30

How many dependent variables do you have (up to five)?
Enter a digit please.    !1

What is the name of the first variable?  !Principal
x = Principal

Give me the name of the cell containing the first item in Principal
!e1

Give me the name of the cell containing the last item in Principal
!e30

Do you want the sum of Principal plotted?  !no
```

If the answer is "yes", another variable is added that is the integral of the curve.

```
    What title would you like?  Please enter it in quotes.

    "Principal Decay"
```

You now receive a plot of your principal array vs your period array, as shown below:

```
            Principal Decay

  45000.00  xx-x-xx---|----|----|----|----|----|----|----|----|
  42999.39  |    |  x x xx x    |    |    |    |    |    |    |
  40998.77  |----|----|----|-xx-x----|----|----|----|----|----|
  38998.16  |    |    |    |    | x xx    |    |    |    |    |
  36997.55  |----|----|----|----|----|-x--|----|----|----|----|
  34996.93  |    |    |    |    |    |   x|x   |    |    |    |
  32996.32  |----|----|----|----|----|----|-x--|----|----|----|
  30995.71  |    |    |    |    |    |    |   x|    |    |    |
  28995.09  |----|----|----|----|----|----|----|x---|----|----|
  26994.48  |    |    |    |    |    |    |    | x  |    |    |
  24993.87  |----|----|----|----|----|----|----|---x|----|----|
  22993.25  |    |    |    |    |    |    |    |    |x   |    |
  20992.64  |----|----|----|----|----|----|----|----|--x-|----|
  18992.03  |    |    |    |    |    |    |    |    |    |    |
  16991.42  |----|----|----|----|----|----|----|----|---x|----|
  14990.80  |    |    |    |    |    |    |    |    |    |    |
  12990.19  |----|----|----|----|----|----|----|----|----|x---|
  10989.58  |    |    |    |    |    |    |    |    |    |    |
   8988.96  |----|----|----|----|----|----|----|----|----|--x-|
   6988.35  |    |    |    |    |    |    |    |    |    |    |
   4987.74  |----|----|----|----|----|----|----|----|----|----x

     1.00000                    (   0.58000)            30.00000

Action:                                           x = Principal
```

You have also entered an interactive plotting environment as described below and you receive the cue "Action" appearing at the lower left of you screen or page. The possible actions you can take at this point are described below the following example.

```
Action:  !hard
Give me a filename:  !Principal
1 request signalled, 45 already in queue 3

Action:  !quit
Do you want to QUIT?  !yes

CELL
```

Entering "hard" in response to the Action prompt causes a print out of a copy of this graph on the printer; "quit" returns you to the MSP environment. Any plotting action (see "Plotting Actions" below) you specify that is not known to the plot environment is taken as a quit, but the system gives you a chance to change your mind.

Plotting Variations

Although the questions and answers are straightforward, there are some variations that might be useful. For example:

- An x increment	If the same cell-name is given for the beginning and the end of the x array, the contents of that cell is used as an increment. The plot starts with x=0 then adds that increment to each succeeding x until you have run out of y values.
- Fixed value of y	If the same cell-name is given for the beginning and the end of a y array, that fixed value is used for each occurrence of that variable. In our example we could have used C1, the payment, in this way and then asked for the sum of the payments. The plot would have looked like the one shown in the example below.
- Cycling	Actually the other two examples are special cases. In the general case, if you have more occurrences of one variable than of others, the plot program variables cycle through the last row in which they appear. It might take some imagination to make use of this feature other than in the case described above. One case would be overlaying curves for recurring periods, such as sales per month where you have the data for several years, just to see if there is a cycle.
- Column arrays	In the example, the arrays lay along one row, but they could as well have taken several rows each or had been by columns. The program uses the elements as they appear in the block defined by the upper left and lower right hand corners. If you said an array started in a1 and ended in c4 the elements would be:

```
a1,a2,a3,a4,b1,b2,b3,b4,c1,c2,c3, and c4.
```

End points a1 and m1 would mean:

```
a1,b1,c1,d1,e1,f1,h1,h1,i1,j1,k1,l1, and m1.
```

Example

```
                Sum of Payments

167593.9 |----|----|----|----|----|----|----|----|----|----*
159493.6 |    |    |    |    |    |    |    |    |    |* * |
151393.2 |----|----|----|----|----|----|----|----|---*|----|
143292.8 |    |    |    |    |    |    |    |    |* * |    |
135192.4 |----|----|----|----|----|----|----|---*|----|----|
127092.1 |    |    |    |    |    |    |    | *  |    |    |
118991.7 |----|----|----|----|----|----|---*|*---|----|----|
110891.3 |    |    |    |    |    |    | *  |    |    |    |
102790.9 |----|----|----|----|----|---*|*---|----|----|----|
 94690.6 |    |    |    |    |    | *  |    |    |    |    |
 86590.2 |----|----|----|----|---**----|----|----|----|----|
 78489.8 |    |    |    |    | *  |    |    |    |    |    |
 70389.5 |----|----|----|--*-*----|----|----|----|----|----|
 62289.1 |    |    |    | *  |    |    |    |    |    |    |
 54188.7 |----|----|--*-*----|----|----|----|----|----|----|
 46088.3 |    |    | *  |    |    |    |    |    |    |    |
 37988.0 |----|----*----|----|----|----|----|----|----|----|
 29887.6 |    |* * |    |    |    |    |    |    |    |    |
 21787.2 |----*----|----|----|----|----|----|----|----|----|
 13686.8 |* * |    |    |    |    |    |    |    |    |    |
  5586.5 *x-x-xx-x-x-xx-x-xx-x-x-xx-x-x|xx-x|xx-x|x-xx|x-x-x
  1.00000                    (   0.58000)          30.00000

Action:                      * = sum_payment      x = payment
```

The Interactive Plot Environment

As mentioned earlier, when you use the MSP $plot command, you enter a new environment. In this environment there are several actions you can take which either modify your graph, cause a hard copy to be produced, or saves the graph for later use. Below are described the most generally useful of these actions, but first a few words of explanation about the plotting itself.

RANGES

The inital x and y ranges are determined by the values provided. The left hand side of the graph represents the minimum x value given, the right hand side the maximum. In the same way, the top and bottom of the graph are defined by the y values (dependent values) provided. These ranges can be changed as described under the change action below.

SYMBOLS

There are a limited number of plotting symbols that can be used to plot a graph. In the above graph, the symbols "x" and "*" are used to plot the x and y coordinates. The plotting symbols used, in order of their selection, are: x, *, +, o, =, ^, %, $, X, and O. If their are more than 10 variables, all of the rest are assigned the symbol "?". The legend at the bottom of each graph explains which symbol has been assigned to which variable.

PLOTTING ACTIONS

save	Save the current plot. Asks you for a name under which it can be saved and later retrieved any time you are in the plot environment. (Actually includes it in a segment called graphs.saved.archive.)
retrieve	Retrieve previously saved plots. If you do not remember the name, type "directory" when asked for a name and a list of the currently saved plots are printed.
hard	Get a hard copy of the graph by producing a segment and directing that segment to the printer.
change	Change values. Usually the only ones changed are ymin,ymax, xmin, and xmax which define the bottom, top, left and right limits respectively. For example:

```
change
Enter new data.  ymin=0, ymax=1000;
```

Note the formal syntax including the terminating semi-colon.

plot	Redisplays the plot.
bestfit	Produces the end points of the least squares line of bestfit for the specified variable. The variable is specified by its symbol and a new symbol is assigned for the end points.

```
Example:  bestfit x X
```

interpolate on	Fills in the broken line.
interpolate off	Removes the filled in symbols.
smooth	Smooths the curves.
normalize y	Sets the y ranges from 0 to 1 and normalizes the curves to this range.

normalize x	Does the same thing for the x range.
range	Gives you the x and y ranges and n, the number of points. Note that each time you do bestfit you add two points. These can cause problems when using interpolate or normalize. To remove, simply reduce n by 2 using change n = digits.

SECTION 19

DEBUGGING FACILITY

MSP provides you with a debugging facility that helps you examine the expressions in your sheet and may be useful when things do not seem to be working the way that they should. It allows you to:

1. check to see which cells, if any, refer to any specific cell. (This feature is called cross-reference and uses the debug command xr.)

2. list all of the expressions contained within any block of cells. (List expressions uses the debug command lx.)

If, for example, changing the value in a cell does not seem to have the effect you expected, you can enter debug and do a cross-reference on that cell to see if it is being referred to as you expected.

THE $DB COMMAND

To enter debug type $db when MSP asks for a cell, just as with any other MSP command. A brief explanation of the debug commands are printed out followed by the symbol [X]. That symbol means that debug is waiting for you to read what has appeared on you screen before it continues displaying information. To continue, type <u>any</u> character followed by a carriage return.

Debug processes each request and then asks for the next command. When you have found your problem, enter qd (quit debug) to return to MSP.

SECTION 20

THE MEMO FACILITY

The Memo Facility lets you produce text files (memorandum, etc.) that incorporate data from your current spreadsheet. The Memo Facility uses your master memo that defines the fixed text, the variable text and the source of the variable data. This master memo is copied with the variable data inserted in the correct location (and format) to produce the final memo. This facility is normally used when the same general outline is used repeatedly, as for a weekly or monthly report.

THE MASTER MEMO

You have to create the Master Memo prior to using the data entry feature of the Facility. The Master Memo contains fixed text, such as, "The gross sales revenue for the period...", which is copied exactly as and where it appears in the Master Memo. It also includes any or all of the control fields described below.

To use the Memo Facility, type the command $memo when the cue "CELL" is displayed. The response displayed is: "Which master memo do you wish to use?" At this point, you have three possible responses:

- "new"
  to go into input mode to create a new master memo
- "edit"
  to change or expand a previously prepared master memo
- "xxx"
  where "xxx" is the name of a previously prepared memo which is to be copied using data from the current spreadsheet.

Each of these is examined below.

Variable Text Control Field

Variable text is text that can change each time the master memo is copied and which is to be supplied by you at that time. The control field is:

```
[[vtext identifier]]
```

where "vtext" is a control keyword, and "identifier" is any string, such as name, address, part no., etc. that you would want used as a cue. When the control field [[vtext *name*]] is found while copying a Master Memo, the program asks what text should be substituted for vtext *name*, as follows:

```
VTEXT name?
```

Type in the necessary information, such as "John J. Jones" and hit the return key. This information is placed in the new memo in place of the control field [[vtext *name*]].

Note that text you type in frequently is longer than the control field. You must consider this when designing your Master Memo.

Label Data Control Field

The label data control field lets you incorporate a row or column label from your spreadsheet into your new memo. The control field form is:

```
[[label rc, column, field width]]
```

where "label" is a control key word, "rc" is replaced by the row or column designation; i.e., a, b, 12, 99, z, etc. "column" is optional and if used tells which column the data field should start in. It must be a number; i.e., 20, 45, etc. "field width" is optional and if used tells how many columns should be used for this field. It is also a number.

Note that labels (and comments) are always left-justified in their allotted space. If neither column or field_width is specified, the label with preceding and trailing blanks removed is substituted for the control string. If field_width is specified but column isn't, the field starts where the control field started.

Cell Data Control Field

The cell data control field lets you incorporate data, numbers and comments, from specified cells in your spreadsheet. The control field form is:

```
[[cell, column, field_width, dec places]]
```

where "cell" is the cell_name, a1, b14, z99, etc. (Note: There is no control key word for this type.) "Column" and "field_width" are the same as label data types (see above) and are both optional. "Dec places" specifies the number of decimal places to be printed for numbers. It is optional and if omitted the currently specified value for the spreadsheet is used.

Note that numbers appear in the form ddd,ddd.dd with commas supplied and leading and trailing blanks removed. The line

```
"The current anticipated revenue of $[[c17]]"
```

assuming c17 contains 1763321.63, would be changed to:

```
"The current anticipated revenue of $1,763,321.63"
```

When field_width is specified, numbers are always right-justified (moved to the right); labels and comments left-justified.

USING RANGES

In Label Data and Cell Data control fields, you can specify a range of cells. For example:

```
[[label a,10]] [[a1,30,12]] [[a2,45,12]]

[[label b,10]] [[b1,30,12]] [[b2,45,12]]
 .            .           .

 .            .           .

 .            .           .

[[label n,10]] [[n1,30,12]] [[n2,45,12]]
```

You want each line to contain the row label and the data from the first two columns for all rows a through n. By using the ranges convention of MSP, all of these lines can be specified by one line:

```
[[label a:n,10]] [[a1:n1,30,12]] [[a2:n2,45,12]]
```

Note that as in MSP, the range is defined by the upper left and lower right-hand corners. Also note that each iteration results in a separate line being produced. If any specified range is shorter than another on the line, the value for the last cell in that range is used until all ranges are exhausted.

CREATING A NEW MASTER MEMO

Typing "new" as the response to the question "Which master memo etc." results in the program going into the input mode. In this mode you can type in your master memo using the fixed text, as you wish it to appear in the final memo, and control fields to handle insertion of the variable data. A short set of instructions is displayed, including the important instruction:

```
"Indicates that you are finished by displaying "input done" as the
first and only characters on a line."
```

After you have finished entering your new master memo and typed "input done" to indicate this, you are asked to provide a name. Please make this a one word or hyphenated name, such as "weekly", "monthly-sales", etc. Your master memo is filed as a Multics segment named "name.master.memo".

EDITING A MASTER MEMO

Once you have created a master memo you can change it, add to it, or delete from it by using the edit facility. Answering "edit" to the question "Which master memo, etc." puts you into the edit mode. You are asked which master memo you want to edit. You are then given a more complete set of instructions. Specifically the instructions are:

```
You must edit from top to bottom in one pass.
```

Commands to find the right line are:

```
'next'.... to go to the next line
'skip n' .. to skip n lines
'print n; .. to print n lines
'line no. n' .. to go to line n
'contains xxxx .. to go to the line that

contains the string xxxx

To change a line type 'c/string1/string2/' where:

you want the first occurance of string1 to be replaced by string2

To delete a line type 'delete'

To add a line type 'add the line to be added'

When you are done type 'done editing'
```

After you have finished editing your master memo, you are asked if you want to replace the old with this new version. If you answer "yes", is it replaced; if you answer "no", you are asked if you want to save this new version; and finally, if you answer "yes" to this, what you want to call it.

USING A MASTER MEMO

If you give any response other than "new" or "edit" to the question "Which master memo, etc.", it is assumed to be the name of a master memo in your files. If the program cannot find a master memo with that name, it lists the names of all the Multics segments having names ending in "master.memo". You are requested to choose one of these or type "quit".

If the master memo is found, you are asked to name the copy. The master memo is then copied, line by line, into a new memo with that name. When the program comes across a control field [[...]], it either extracts the data from your current spreadsheet or, in the case of vtext, asks you to supply the needed information.

When the end of the master memo is reached, you are given the opportunity to review the finished memo. You are then asked if you want it routed to the printer. If you reply "no", the name of your new Multics segment is displayed. You could, presumably, run this segment through a text formatter, such as COMPOSE, to produce a more finished product.

MEMO ARGUMENTS

When you are producing your master memo and you encounter data other than vtext that you want to leave undefined until you actually use the memo, you can replace those items in the master memo with a memo argument field. These fields have the form "&&1", "&&2", "&&3"..."&&9". When the memo facility encounters one of these forms in a line on the master memo, it asks you to declare the value by typing:

```
&&n? (where n is the argument number 1 to 9)
```

Reply by typing the string which is to replace that argument wherever it appears in the master memo and hitting return. Once a particular argument is defined, that definition is used throughout the memo without any further prompts.

The following is an example of using memo arguments:

```
The values for the last five weeks are:
[[label &&1:&&2,10,20]][[&&16:&&26,30,12]]
```

If when running this memo you replied:

```
&&1?  c
&&2?  g
```

the lines would be interpreted as:

```
The values for the last five weeks are:
[[label c:g,10,20]][[c6:g6,30,12]]
```

Example

This example of a master memo uses the mortgage example spreadsheet described in Section 14.

EXAMPLE OF USING THE MEMO FACILITY

Mortgage Example

```
Loan applicant:  [[vtext name]]

                 [[vtext street]]

                 [[vtext city]]

Location of property:  [[vtext location]]

Description of property:  [[vtext description]]

Assessed value:  $[[a2]]

Amount of loan:  $[[a1]]

Terms:  [[b2,,,0]] years at [[b1,,,3]]

Annual payment:  $[[b3]]

Equity after &&1 years is $[[d&&1]]
```

Having produced this master memo and calling it "example", you'd use it as follows:

```
CELL  $memo

Which master memo should I use?  example
VTEXT name?  Jack and Jill Jones
VTEXT street?  22 Pleasant Street
VTEXT city?  Pleasantville, MM 11111
VTEXT location?  31 East Street, Pleasantville
VTEXT description?  single family brick dwelling
&&1?  10
END OF example MEMO REACHED
Do you want to review your memo?  yes

Loan applicant:     Jack and Jill Jones
                    22 Pleasant Street
                    Pleasantville, MM 11111

Location of property:  31 Easy Street, Pleasantville

Description of property:  single family brick dwelling

Assessed value:  $84,000.00

Terms:  30 years at 0.125

Annual payment:  $5,794.21

Equity after 10 years is $42,042.10

Do you want your memo printed?  yes

1 request submitted, etc.

CELL
```

APPENDIX A

THE FILING FACILITY TO SPREADSHEET CONNECTION

At times, it might be convenient to create an MSP file using data from a card file. The Filing Facility to spreadsheet connection lets you invoke MSP, load your newly created MSP file with data from a card file, do your analysis, and then return to the Multics simplified filing facility.

TYPES OF TRANSFORMATIONS

There are two different formats for moving data from a card file to an MSP pad. They are:

- List - Each item in the card file that is moved occupies one row or column in the MSP file.
- Table - Items in the card file can be arranged in tabular form in the SP file.

Before creating an MSP file, choose the appropriate method for the task. Unfortunately, they cannot be mixed.

The List Transformation

The List transformation method moves each item in the card file into one row or column in the MSP file. Assume that the items in our card file constitute the orders for one line of products for one month. The items are order no., item name, cost, no. sold, and price. You want to analyze the effect of raising the unit price 10% while offering various quantity discounts. In this case, you want to move the items into MSP and allow each item occupy a separate row. Also, in this case, the items "order no." and "item name" should be entered as MSP comments and the items cost, no. sold, and price to be entered as numerical values (numbers). The item names would be assigned as labels to the corresponding row or column.

The Table Transformation

This Table transformation method lets you arrange the values of one item (the target item) in tabular form, depending on the values of two other items, one of which determines the row and the other the column where the target item is to be placed. For example, assume our file contains model no., color and total units sold for a product and you are interested in determining if color preference varies between the different models. (Do people who choose luxury models of automobiles prefer black?) In this case, you want to arrange our pad so that total units sold for each color appear down one column (or across one row) and the total units sold for each model no. appear across one row (or down one column). You also have the choice of entering the model no. and color as comments in a cell or as a row and column labels, depending on where in the pad you wish to place it and what you are doing.

CREATING A SPREADSHEET FILE

The command to create a MSP file is

```
create a spreadsheet file
```

The first response you receive from MSP is

```
WHAT WILL YOU CALL THIS PAD?
```

Respond with a single word name, such as "sales" or a multi word name with the words joined by underscores, such as october_sales. This is the name you will use later to load the file into MSP using the $load command.

You are then asked:

```
WILL THIS BE A LIST OR TABLE TYPE PROCESS?
```

Depending on your response, either "list" or "table", enter one of the two processes described below.

The List Process

To generate the list form of an MSP file, you must specify the following information about each item to be passed to the file:

- The item name
- The row or column it will occupy. (It is assumed that each row item starts in column 1 and each column item in row a.)
- Whether it should be considered as alphanumeric information and passed as a comment, or numeric information and passed as a value.

The syntax used is:

item name (pos, form), item name (pos, form), ...

where *item name* is the name of the item being passed and *pos* is a number (for column number) or a lower case letter (for row name) indicating which col or row this item will occupy. (Although, theoretically one item in the file can be inserted into a column and another into a row.) The *form* is either "a" or "n" denoting alphanumeric or number.

When you invoke the list process, this syntax is explained to you again.

The item names that you specify are used to label the appropriate rows or columns (notice this when you load the MSP file.)

Whether you should insert items across in rows or down in columns depends primarily on how many cards you have in your subset. In no case can MSP handle more then 26 rows or 99 columns.

The Table Process

If you invoke the table process, the first question is

```
WHICH ITEM WILL CORRESPOND TO THE ROW INDEX?
```

You should generally reply with the name of the item having fewer variations.

If an item has more variations than are able to fit in the space allotted, the latter are dropped and corresponding target values do not appear in the table.

The next question is

```
SHOULD THE VALUES IN item name BE USED AS ROW LABELS?
```

By answering "yes" to this question you take advantage of the labeling capability of MSP. You might want these "labels" to appear elsewhere on the pad, however. In this case, answer "no" and the response is:

```
THEN WHAT COLUMN DO YOU WANT USED?
```

Specify a number, from 1 to 99 that positions the table in the horizontal scale.

Note that the row and column indices form upper and left-hand edges for the table.

You are then asked a similar set of questions about the column index. At that point, the program runs through your card file and generates the two sets of indices by arranging the values it finds in alphabetic order. It then reports on how many row and column indices have been generated and asks

```
WHICH ITEM WILL YOU USE TO FILL THE TABLE?
```

Respond with the name of the target item.

You are then asked:

```
IS THAT A number OR A comment?
```

Respond "number" (usually) or "comment", and the program proceeds to fill in the table and generate your MSP file. When finished, it prints out "end of file name" and returns the > prompt.

INVOKING MSP

You can invoke MSP by simply typing

```
spreadsheet
```

Assuming that you have read the MSP description and made the necessary links in your directory.

as a command. You receive the response

```
cell_name
```

which is MSP waiting for your next command (usually $load file_name). When you finish with MSP and type $quit, you are returned to the Multics simplified filing facility and receive the prompt. You can, of course, invoke MSP outside of the the Multics simplified filing facility environment and load the files you generated with filing facility, since they are simply MSP saved files.

APPENDIX B

COMMAND SUMMARY

The following is an alphabetical list of the Multics simplified filing system commands. Note that words in italics indicate variable portions of the command line and underlined words can be entered as a short for of the command line.

```
change item name to name
change item name(s) to name
change the name of item name to item name
change to file file name
change to file file name
choose from file name on item(s)
combine with file name on item name(s)
copy item name changing item name to value
copy item name setting item name to expression
copy all cards n times
copy this card n times
couple with file name
create a new file
decode item name key is key name
delete this card
difference from file name on item name
difference from file name on item name
display the active file information
display the active file information
display the open file information
encode item name key is key name
enter n cards
enter a card
evaluate expression
fill in item name
fill in item name on n cards
enter n cards
fill in item name on n cards keying
  on item name(s).
find the card where
find the next card where
find the next n cards where
find those cards where
fix decimal places to n
globally change item name to name
globally change item name(s) to name
go to card n
if conditional expression then action
  else action
index of item name
index of item name where item name contains name
insert a card
```

insert an item *item name* before *item name*

labels list

let *item name* = *expression*

move card n before (or after) this card (or card n)

move this card before (or after) card n

open the file

print

print *item name(s)* break on *item name(s)*

print *item name(s)(col_start,col_width)*

print *item name(s)(line_no,col_start,col_width)*

print *item name(s)*

read *segment name* into *file name*

remove the item *item name*

reset the sequence

restore the sequence

retain _______ of *item name* by *item name*

retain this subset

run *file name*

save this file

save this subset

select those cards where

set *item name* to *item name*

set *item name* to *expression* converted to *type*

set *item name* to *expression* converted from *type* to *type*

show me all cards

show me this card

show me the first card

show me the last card

show me the next *n* cards

show me the next card

show me the previous *n* cards

show me the previous card

skip back *n* cards

skip forward *n* cards

skip to the first card

skip to the last card

sort by date on *item name*

sort by table on *item name*

sort by value on *item name*

sort on *item name*

sum of *item name*

sum of *item name* by *item name(s)*

sum of expression

sum of expression by *item name*

INDEX

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	MULTICS SIMPLIFIED COMPUTING AND FILING FACILITY	ORDER NO.	GL71-00
		DATED	FEBRUARY 1984

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell

40200, 7.5C584, Printed in U.S.A.

GL71-00