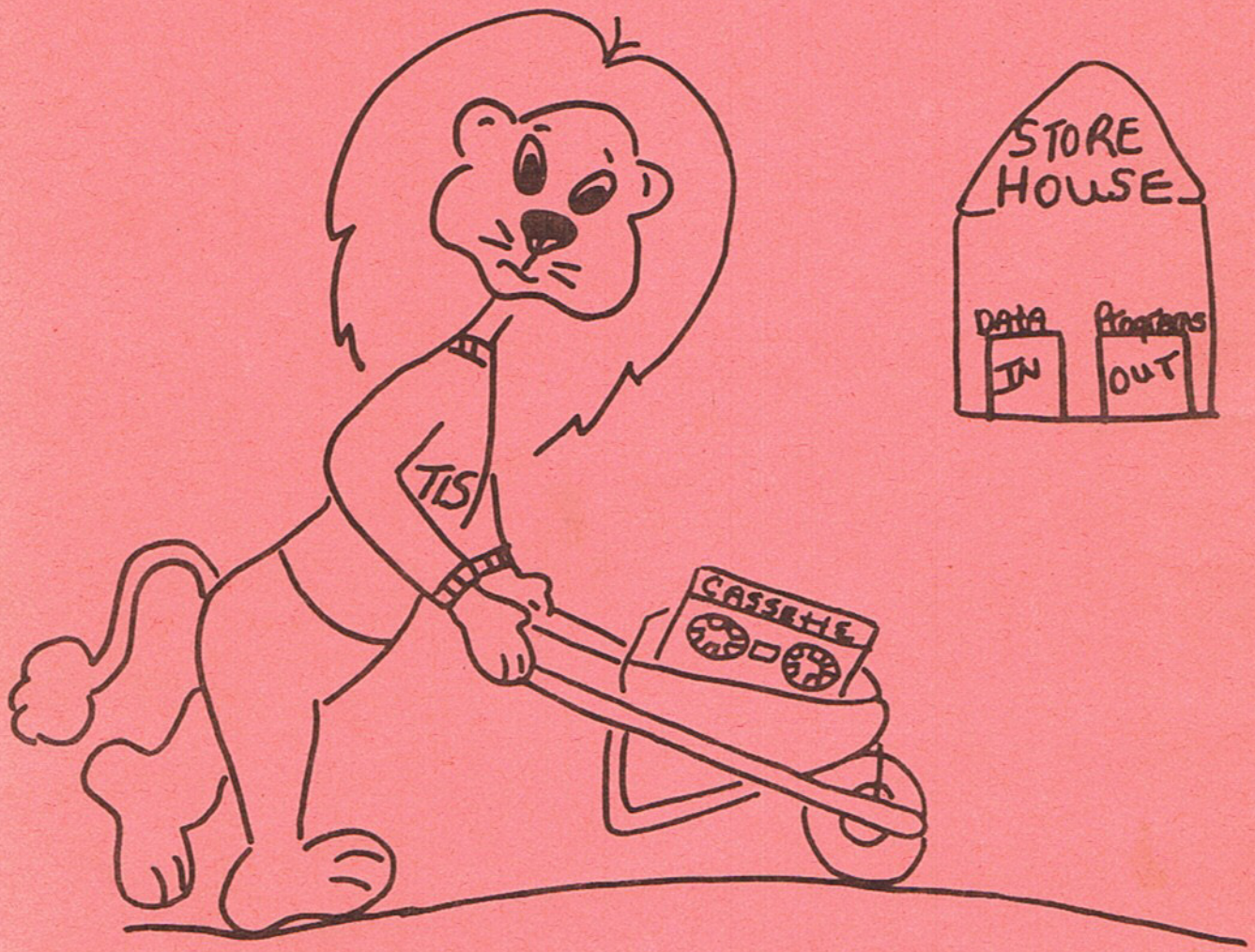
# TIS Workbook 4

TOTAL INFORMATION SERVICES




PET Cassette

CONTENTS

CASSETTE STORAGE

# I. INTRODUCTION

This workbook gives a series of exercises that show the user how to use the cassette storage system on the Commodore PET 2000 computer. The most effective way to use the workbook is to sit down with a PET and do the exercises as they are presented. Enough space has been provided in the workbook for you to add your own examples as you develop them. Later, when you need to refresh your memory on a particular topic, these examples will supply pertinent, meaningful information.

## A. Notation Used in This Workbook

We use a consistent notation in this workbook to indicate what is to be typed on the keyboard (T:), what appears on the TV display (R:), and what indicates blanks are to be typed (b). For example:

```
T: info ('RETURN' key)
```

means to type the characters contained on the line after the colon (:) followed by a 'RETURN'.

```
R: response
```

means that the system response to the previous T: line should be a line on the TV.

Blanks are important. They are specified by b. For example:

```
T: ?"ABbC" ('RETURN' key)
```

means type the characters ?", then the letter A, then the letter B, then a space, then the letter C, then " followed by a 'RETURN'.

Now let's run that example all together:

```
T: ?"ABbC" ('RETURN' key)
R: AB C
```

The 'RETURN' key must be pressed at the end of each line. We will assume you now know that and will not use ('RETURN' key) in any more examples.

The special keys on the PET keyboard can cause some confusion. This workbook identifies the special keys with the notation 'KEYNAME'. 'KEYNAME' means press the named key. The unquoted sequence of characters OTHER means, press the five keys O T H E R in succession.

Example: Special key notation

T: 'STOP'

means press the key labelled STOP

T: STOP
R: BREAK
R: READY.

means press the four keys S T O P in succession.

B. Exercises

Some of the exercises are designed to be done by rote; that is, if you type this, then you should get this response. As you learn more about a feature, the exercises take the form "What would you get if you typed this?" The next level suggests that you try various type-ins.

If you understand the topic, you should be able to anticipate most of the responses. In cases where there are idiosyncrasies, exceptions, and special problems, we have written "try-these" exercises.

The last level of exercises is "do-it-yourself," in which you are given guidelines to create a personal example. If you find you cannot construct a personal example, review the previous material and try again.

You will find that you will learn more if you take the time to create or try new things. Be sure to enter the results of both successful and unsuccessful attempts in your workbook so you have a convenient reference about what you have done.

Do all the exercises in the workbook. DON'T skip any of them.

C. Program Storage

The Commodore PET uses a modified audio cassette for program and

data storage. A program can be written on the cassette with the SAVE command. Later, when you wish to use the program again, you can read it back into memory with the LOAD command. Information from a program (data) can be written on the cassette with the PRINT# statement. Later when you wish to use the data again, you can read it back into another program with the INPUT # or GET # statement.

Data written with the SAVE command cannot be read with INPUT # or GET #. Likewise, data written with PRINT# cannot be read with LOAD.

Program storage is covered first so that we can build a set of subroutines to manipulate data files. By saving these programs on cassette we can reuse them later. This will save typing them in repeatedly.

1. SAVE a Program

SAVE"LABEL"

To write a program on the cassette:

```
T:  SAVE
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK - this occurs after you press play and record
R:  WRITING
R:  READY.
```

If you wish to give your program a label (name), type SAVE"LABEL". "LABEL" may be from 1-74 characters long.

To write a program on the tape with the label ABC:

```
T:  SAVE"ABC"
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK - this occurs after you press play and record
R:  WRITING ABC
R:  READY.
```

WARNING!WARNING! The system does not check to see which buttons are pushed on the cassette. If any button is left on, the system assumes it is the right one. Whatever button you push is assumed to be the correct one.

2. VERIFY a Program

VERIFY"LABEL"

To be confident that the program you wrote on the cassette is written correctly, you should compare what is on the tape with what is in memory by the VERIFY command. Rewind the cassette then:

Exercise: Verify that the information just written is correct.

```
T:  VERIFY
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING
R:  FOUND
R:  VERIFYING
R:  OK
R:  READY.
```

If there is a compare error, you will get a ?VERIFY ERROR. If this happens, rewind the tape and try again. Since cassette reading and writing is not all that reliable, trying again often gets past the problem.

If a VERIFY is typed without a label, the system will try to compare the first program it finds on the cassette with what is in memory. If a VERIFY"LABEL" is typed, the system will search for the program with the given label. The name of each program before the label will be listed.

Cassettes may be removed without rewinding, then replaced, and a new program written on the cassette from that point. A 10-15 second leader is written after each SAVE command is given.

Exercise: SAVE a program and demonstrate that VERIFY catches differences.

```
T:  NEW
T:  10 PRINT"TAPE TEST"
T:  20 INPUT L,W
T:  30 A=L*W
T:  40 PRINT A,L,W
T:  50 END
```

Rewind the cassette.

```
T:  SAVE"A"
R:  PRESS PLAY AND RECORD ON TAPE #1
R:  OK
R:  WRITING A
R:  READY.
```

Rewind the cassette.

```
T:  VERIFY
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING
R:  FOUND A
R:  VERIFYING
R:  OK
R:  READY.
```

Turn "PLAY" off before saving ABC.

```
T:  SAVE"ABC"
R:  PRESS PLAY AND RECORD ON TAPE #1
R:  OK
R:  WRITING ABC
R:  READ.
```

Rewind the cassette.

```
T:  40
T:  45 PRINT A,L,W
```

Change line number 40 to 45.

```
T:  VERIFY"ABC"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR ABC
R:  FOUND A
R:  FOUND ABC
R:  VERIFYING
R:  ? VERIFY ERROR
R:  READY.
```

### 3. LOAD a Program

To reload your program rewind the cassette and type

```
T:  LOAD
R:  PRESS PLAY ON TAPE #1
```

The LOAD command automatically clears memory and reads into memory the first program on the cassette. The following responses are displayed after you press play on the recorder.

```
R:  OK
R:  SEARCHING
R:  FOUND A
R:  LOADING
R:  READY.
```

If you want to load a labeled program, rewind the cassette and type LOAD"A".

Exercise: LOADing of programs SAVEd previously.

```
T:  LOAD"A"
R:  PRESS PLAY ON TAPE #1
R:  SEARCHING FOR A
R:  FOUND A
R:  LOADING
R:  READY.
```

When LOAD is typed, the first program found is the one loaded. For example, if you have three programs on a cassette and the recorder is sitting at the end of the first program, LOAD will load the second program.

If a LOAD"ABC" is given, a search will be made for program ABC. When ABC is found it will be loaded. A list of all programs found before ABC will be displayed. For example, suppose there are four programs on a cassette, the first two are labeled 1 and $, respectively; the third is not labeled; and the fourth is labeled PET.

If a LOAD"PET" is given, the system responds as follows:

```
T:  LOAD"PET"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR PET
R:  FOUND 1
R:  FOUND $
R:  FOUND
R:  FOUND PET
R:  LOADING
R:  READY.
```

If you do not know what is on a cassette, just ask to LOAD a non-existent label and the name of each program on the cassette will be listed as it is found.

4. Improving Your Chances of Success. Loading a program from tape is not the most reliable operation in the world. The most common error is that the system finds the label it is searching for but never successfully loads the program. The only corrective measure is to rewind the cassette and try again. Often, a second or third try will be successful.

   If you have a tape that you must read and retrying doesn't help, cleaning the cassette recorder head with a good solvent or with a non-abrasive cleaning tape has helped on occasion. Once you get the program loaded, save it on another tape.

   When you SAVE a program you can specify whether a single end of file (EOF) or an EOF and an end of tape (EOT) are to be written. The default (which we have used in the previous examples) is a single EOF.

   If you SAVE"FILENAME", 1,2 the 1 is the device number of the standard built-in cassette and the 2 specifies EOT. When you use this option, you must be careful how you write additional information on this tape.

   To write additional information you must rewind the tape and VERIFY the FILENAME. VERIFY leaves the cassette positioned after the SAVEd file but before the EOT. When you SAVE another file on that tape you will write over the first EOT.

   If you fail to do the VERIFY step before SAVEing another program, it will be written after the EOT and it will be difficult (but not impossible) to recover.

   Why use the EOT option if it can cause you problems? The EOT option can save you time when searching a tape. The cassette

software will give you a message when the EOT signal is encountered. That can save you a long wait if you are searching a tape with nothing more on it.

Exercise: Use EOT option on SAVE. Rewind a cassette.

```
T:  NEW
T:  100 PRINT "FIRST SAVE FILE"
T:  SAVE"FIRST",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING FIRST
R:  READY.
```

This writes a copy of the program followed by an EOF signal and then an EOT signal. The recorder is now positioned after the EOT signal.

Exercise: Write a second program after the EOT and try to recover it.

```
T:  100 PRINT "SECOND SAVE FILE"
T:  SAVE"SECOND",1,2
R:  WRITING SECOND
```

Now rewind Tape #1 and try to load SECOND.

```
T:  LOAD"SECOND"
R:  PRESS PLAY ON TAPE #1
R:  OK

R:  SEARCHING FOR SECOND
R:  FOUND FIRST
R:  ?FILE NOT FOUND ERROR
R:  READY.
```

The ?FILE NOT FOUND message means that in searching, the load routine found an EOT before it found the name specified (SECOND). The cassette is now positioned after the first EOT.

```
T:  LOAD"SECOND"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR SECOND
R:  FOUND SECOND
R:  LOADING
R:  READY.
T:  LIST
R:  100 PRINT "SECOND SAVE FILE"
R:  READY.
```

This shows that you can load a program that is after the EOT, if you read past the EOT.

Exercise: Position tape so that second file is written over the first EOT. First rewind the cassette.

```
T:  LOAD
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING
R:  FOUND FIRST
R:  LOADING
R:  READY.
T:  LIST
R:  100 PRINT "FIRST SAVE FILE"
R:  READY.
```

Rewind the cassette so that we can rewrite the first file and show how to get positioned properly before the second write.

```
T:  SAVE"FIRST",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING FIRST
R:  READY.
```

Now rewind cassette #1.

```
T:  VERIFY "FIRST"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR FIRST
R:  FOUND FIRST
R:  VERIFYING
R:  OK
R:  READY.
```

The cassette is now positioned after the FIRST program and before the EOT signal.

```
T:  100 PRINT "SECOND SAVE FILE"
T:  SAVE"SECOND",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING SECOND
R:  READY.
```

The copy of the SECOND program is written over the first EOT. Now rewind cassette #1.

```
T:  LOAD"SECOND"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR SECOND
R:  FOUND FIRST
R:  FOUND SECOND
R:  LOADING
R:  READY.
```

The benefit of using the EOT option is that you will not spend as much time searching for a program that is not on the tape. Assume that you forgot that SECOND was the last program on tape and you wanted to load the next one.

```
T:  LOAD
R:  SEARCHING
```

A short time later the message

R: ?FILE NOT FOUND ERROR

There is no need to guess how long you should wait for a FOUND message.

D. Cassette Characteristics

The Commodore PET uses a modified audio cassette for program and data storage. Information is recorded on cassette at an effective rate of about 50 bytes per second. See Section V for a detailed determination of data rate.

1. Data Transfer. To better understand how the PET cassette software works, let's look at the steps taken to get information from your program to a cassette and back.

a. Write steps:

Step 1. - Data is moved from the program to a special area of memory called the cassette buffer. There are two cassette buffers. The buffer for cassette #1 (the standard built-in cassette) is the 191 bytes of memory from 27B to 339 hexidecimal (635 to 825 decimal). The buffer for cassette #2 (the extra cassette) is the 191 bytes of memory from 33B to 3F9 (827 to 1017 decimal). The statement used to move data to buffer one (635-825) is PRINT#1 PRINT#2 uses the memory area from 827 to 1017.

Step 2. - When the cassette buffer is full, the data is moved from the buffer to the tape. This is also done when CLOSE is used. See Fig. 1

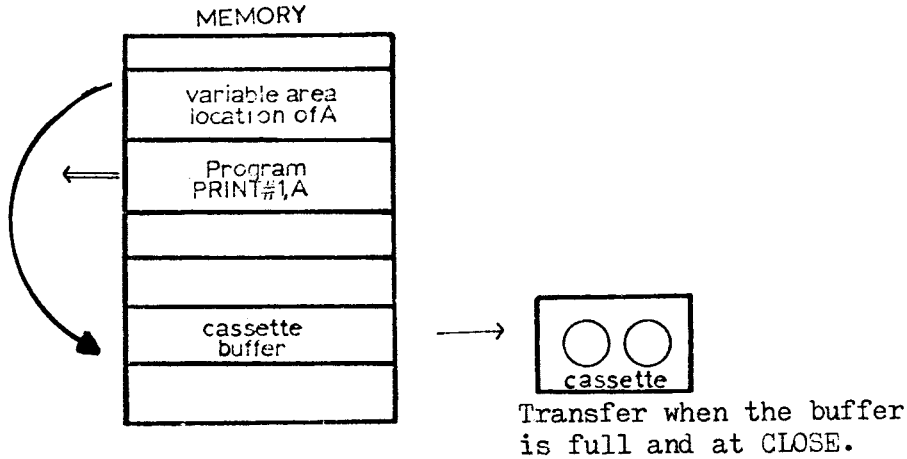



Fig. 1. Data transfer - PRINT#

b. Read steps:

Step 1. - Data is moved from the tape to the cassette buffer.

Step 2. - Data is moved from the cassette buffer to the program. The statements used to move data from the buffer to your variables are INPUT# and GET#. Fig. 2 shows the path that data follows.

2. File Format. Information is written from the cassette buffer to the tape in blocks of 191 bytes. Data is recorded on tape just like data is written to the TV display. When a file is CLOSED, the PET always writes an end of file (EOF) signal. You can tell the PET to write an end of tape (EOT) after the EOF. You can see in Fig. 3 the logical cassette layout.

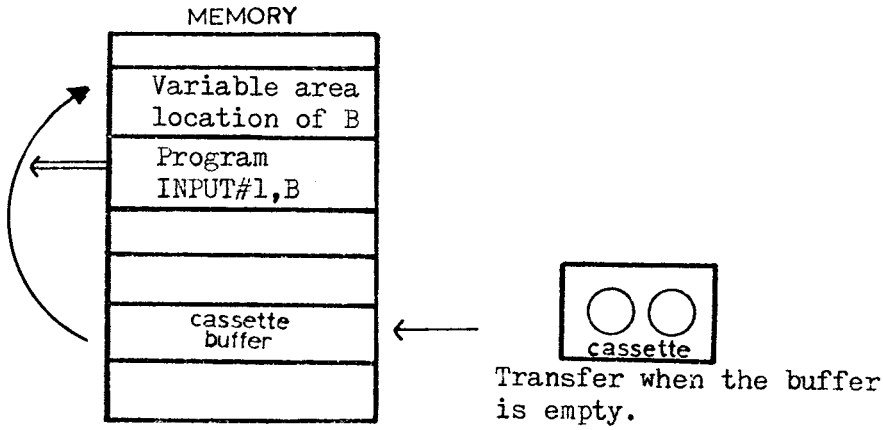



Fig. 2. Data transfer - INPUT#

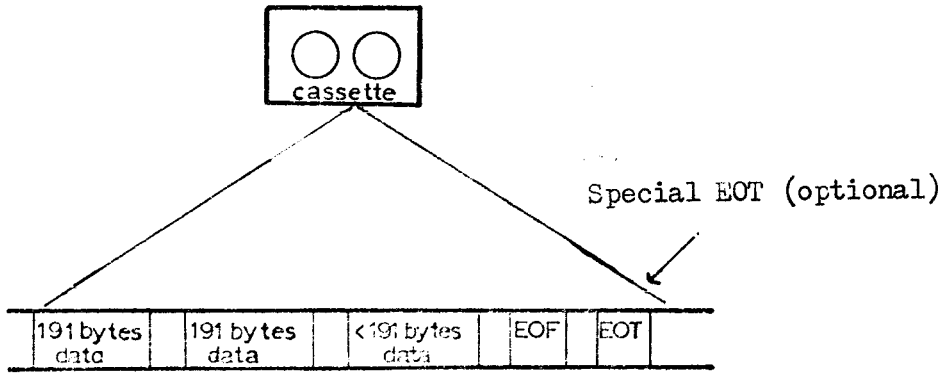



Fig. 3. File layout.

3. File Name Conventions. File names can be made of any legal printable character. The maximum length of a file name is reported to be 187 characters. Only the first 16 characters are displayed in file search and file found messages.

   Exercise: Determine the maximum file name length that can be displayed in cassette messages. Rewind the cassette.

```
T:  NEW
T:  100 FOR I=1 TO 20
T:  110 NA$=NA$+CHR$(64+I)
T:  120 NEXT I
T:  130 OPEN 1,1,2,NA$
T:  140 CLOSE 1
T:  RUN

R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  READY.
```

Rewind the cassette.

```
T:  LOAD
R:  PRESS PLAY ON TAPE #1
R:  SEARCHING
R:  FOUND ABCDEFGHIJKLMNOP
R:  ?FILE NOT FOUND ERROR
R:  READY.
```

Even though NA$ had 20 characters only the first 16 were displayed in the tape message. To convince yourself that NA$ has more letters than displayed:

```
T:  PRINT NA$
R:  ABCDEFGHIJKLMNOPQRST
R:  READY.
T:  PRINT LEN(NA$)
R:  20
R:  READY.
```

File name searches are done on the basis of first character matches. That means that a short name like NOW will "match" file names like NOWISTHE or NOWHERE. No attempt is made to see if the name found is longer than the file specified.

Exercise: Demonstrate file name search strategy.
Rewind cassette #1.

```
T:  NEW
T:  100 PRINT "KNOWN"
T:  SAVE "KNOWN"
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING KNOWN
R:  READY.
```

This puts the program KNOWN out as the first program on tape.

```
T:  NEW
T:  100 PRINT "NOWHERE"
T:  SAVE "NOWHERE"
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING NOWHERE
R:  READY.
```

The second program is called NOWHERE.

```
T:  NEW
T:  100 PRINT "NOW"
T:  SAVE "NOW",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING NOW
R:  READY.
```

The last program is NOW and it is followed by an EOF and an EOT. Rewind the cassette.

```
T:  LOAD "NOW"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR NOW
R:  FOUND KNOWN
R:  FOUND NOWHERE
R:  LOADING
R:  READY.
T:  LIST

R:  100 PRINT "NOWHERE"
R:  READY.
```

File named NOWHERE was loaded instead of NOW. Rewind the cassette.

```
T:  LOAD "KNO"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR KNO
R:  FOUND KNOWN
R:  LOADING
R:  READY.
```

The search strategy considered KNO and KNOWN to match!

Exercise: Further exploration of the effect of matching substrings. Rewind the cassette.

```
T:  NEW
T:  100 PRINT "REMODEL"
T:  SAVE "REMODEL"
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING REMODEL
R:  READY.
T:  100 PRINT "MODE"
T:  SAVE "MODE",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING MODE
R:  READY.
```

We now have two programs on this cassette. REMODEL is the first one and MODE is the second one. Rewind the cassette.

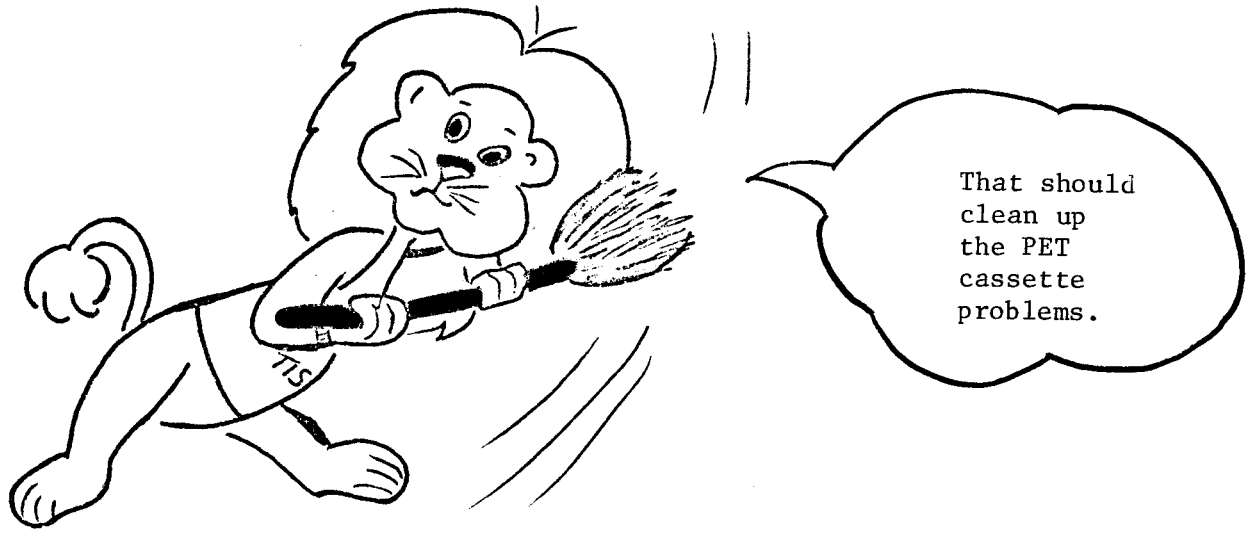
```
T:  LOAD "MODE"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR MODE
R:  FOUND REMODEL
R:  FOUND MODE
R:  LOADING
R:  READY.
T:  RUN
R:  MODE
R:  READY.
```

This shows that matching substrings only affect the LOAD command, if they start at the beginning of the file name.

4. Timing Problems. Early versions of the PET have some problems with the cassette software due to timing. There needs to be more time (space) before the file name record, between data files, and between data records.

   When you work with the PRINT# statement, you will develop a program that makes sure that there is enough time between data records. That program checks to see if a data record was just written on tape. A .1 second gap is written on the cassette after each data record.

   The space before the file name record can be improved by stopping the cassette before you open a file. The PET will tell you to make the cassette ready with the message PRESS PLAY & RECORD ON TAPE#1. Do not leave the play and record button on when you go to open a write file. Files written after you respond to the prompt and the PET reply OK seem to be more reliable than other files.



## II. DATA FILE HANDLING

There are three (3) steps needed to successfully write data from a program to the cassette or to read data from a cassette and use in your program. The first step is to tell the PET operating system software that you intend to write (or read) data to (from) the cassette. The statement to do that is OPEN.

The second step is to actually do the data transfer, i.e., read or write. The OPEN in the first step must match the data transfer direction. To write data to cassette the single statement PRINT# is used. To read a data cassette either an INPUT# or a GET# statement can be used. We will explore the differences between these two input statements later.

The third and final step is to tell the operating system that you are finished with cassette input or output. On a cassette that was opened for a write, any information still in the cassette buffer is written to the tape along with the specified trailing special blocks. Remember that the system transfers data automatically from the cassette buffer to the the tape only when the cassette buffer is full. Any partial buffer remaining must be sent to tape with the CLOSE statement.

### A. OPEN

The OPEN statement prepares a file for either input or output. There are four possible parameters, each separated by commas. The form of the statement is OPEN A,B,C,D$.

#### 1. Parameters.

a. Logical file number:

The first parameter (A) is the logical file number. It is the only parameter that is required. The logical file number can be a constant or a numeric variable whose value is between 1 and 255. All other input or output statements must refer to this file with this logical file number.

b. Physical device number:

The second parameter (B) specifies the physical device number. A physical device number of 1 refers to the standard built-in cassette and a number of 2 refers to the auxiliary cassette.

c. Data direction:

The third parameter (C) specifies the input or output

option. There are three legal values:

0 means open for a read (input)
1 means open for a write (output). On close only an end of file (EOF) marker will be written after the last (partial) data block.
2 means open for a write (output). On close an EOF marker will be written after the last (partial) data block and an end of tape (EOT) marker will be written after the EOF.

d. File name:

The last parameter (D$) is the file name. The maximum length of a file name is reported to be 187 characters.

2. Default Values. There is no default for the first parameter. You must specify the logical file number. If you specify a parameter, you must specify all the parameters to the left of it, i.e., all that precede it on the line. It is illegal to specify OPEN A,,C,D$.

If you do not specify a parameter, these defaults are used:

Physical device number (B) = 1
Input or output option (C) = 0 i.e., read
File name (D$) = "" i.e., null (matches every file name)

3. Experiments with OPEN.

Exercise: Demonstrate the results of different OPENs.

```
T:  NEW
T:  A=1
T:  OPEN A
R:  PRESS PLAY ON TAPE #1
T:  'STOP'
R:  BREAK
R:  READY.
```

A filename of 1 is OK. The 'STOP' key is used to terminate the attempted OPEN. Do not press play on tape #1. We just want to see if your PET will accept the OPEN.

```
T:  NEW
T:  A=0
T:  OPEN A
R:  ?SYNTAX ERROR
R:  READY.
```

Zero is not a legal file number for an OPEN.

```
T:  NEW
T:  A=-1
T:  OPEN A
R:  ?ILLEGAL QUANTITY ERROR
```

Negative numbers are not legal as a file number. Therefore, one (1) is the smallest legal file number.

```
T:  NEW
T:  A=255
T:  OPEN A
R:  PRESS PLAY ON TAPE #1
T:  'STOP'
R:  BREAK
R:  READY
```

A file number of 255 is OK.

```
T:  NEW
T:  A=256
T:  OPEN A
R:  ?ILLEGAL QUANTITY ERROR
```

Since 256 is too big a file number, 255 is the largest legal file number. What would you expect to happen if you used an integer variable like A%? Try a few.

Doen

```
T:
T:
T:
R:
R:
R:
```

Werkt ook met decimale getallen tussen 0 en 256

T:

T:

T:

R:

R:

R:

Integer variables between 1 and 255 are legal for OPEN file numbers.

```
T:  NEW
T:  A$="1"
T:  OPEN A$
R:  ?TYPE MISMATCH ERROR
```

The variable must be numeric. A string variable is illegal for the file number in an OPEN.

```
T:  NEW
T:  OPEN 1,1,256
R:  ?ILLEGAL QUANTITY ERROR
```

The value must be less than 256 to be a legal value for the input or output option. Any value greater than 1 is treated as a 2!

```
T:  NEW
T:  OPEN 1,1,-1
R:  ?ILLEGAL QUANTITY ERROR
```

A negative number is not a legal value for the input or output options.

```
T:  NEW
T:  OPEN 1,1,-0
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING
T:  'STOP'
```

Since -0 = 0 this open works as an open for read.

B. Data Transfer

The direction of data transfer must match the direction that the OPEN specified.

1. PRINT#. The output statement (PRINT#) specifies the logical file number followed by a comma and the variable list.

   Example: PRINT#1,A

   The logical file number must match a logical file number in a previous OPEN with the output option specified, i.e., OPEN 1,1,1

   PRINT# writes to cassette the same characters that would appear on the TV display if you used PRINT with the same variable list. This means that not all 256 characters can be written on cassette. There are no errors detected on a PRINT# statement. That is, you can write anything that you want on cassette. However, you must choose carefully what you write so that you can read it back later.

   If you write more than one variable per line, you must include on the cassette information to separate the variables.

   a. Writing numbers:

      The PRINT# will handle numeric variables just as the regular PRINT to the TV.

      Exercise: Demonstrate write and read back with one numeric variable for each PRINT#.

```
T:  NEW
T:  100 OPEN 1,1,2, "TEST"
```

This opens logical file number 1 on physical device number 1 (the built-in cassette) for output. An EOF and an EOT will be written when the CLOSE 1 statement is executed. The file name is "TEST".

```
T:  110 FOR I=1 TO 5
T:  120 PRINT #1,I
T:  130 NEXT I
T:  140 CLOSE 1
```

Write 5 lines with the number 1 to 5 in the lines. Then close the file with an EOF and an EOT. Note that only a small amount of data was written. This simple program will not work with a large amount of data. A program that will handle large files will be described later.

```
T:  150 PRINT "REWIND TAPE--THEN PRESS ANY KEY"
T:  160 GET A$:IFA$="" THEN 160
T:  170 OPEN 1,1,0, "TEST"
```

After rewinding the tape, open logical file number 1 on device 1 for a read. The file name to search for "TEST".

```
T:  180 FOR I=1 TO 5
T:  190 INPUT# 1,X
T:  200 PRINT X
T:  210 NEXT I
```

Read back each value written on tape and display the data read on the TV.

```
T:  220 PRINT "DONE"
T:  230 CLOSE 1
```

Lets you know that the program is done and then tells the PET that we are finished with the open file.

To save typing you should save this program for use later. Call it SINGLE. Rewind a program cassette.

```
T:  SAVE "SINGLE",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING SINGLE
R:  READY.
```

Rewind cassette #1.

```
T:  VERIFY "SINGLE"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR SINGLE
R:  FOUND SINGLE
R:  VERIFYING
R:  OK
R:  READY.
```

Unload the cassette and mark it SINGLE. Load a scratch cassette and rewind it.

```
T:  RUN
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  1
R:  2
R:  3
R:  4
R:  5
R:  DONE
R:  READY.
```

Let us change the program in the last exercise to handle two variables at a time. The straight forward approach is to simply add a list variable to the PRINT#1 and the INPUT#1.

Exercise: Experiment with writing and reading two values at a time. If you have turned off the power or typed NEW since entering the last exercise, reload SINGLE by rewinding the tape and

```
T:  LOAD "SINGLE"
T:  120 PRINT#1,I;I*I

T:  190 INPUT#1,X,Y
T:  200 PRINT X,Y
```

These lines modify the original exercise to write and read two values at a time. Load a scratch cassette and rewind it.

```
T:  RUN
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  BAD DATA ERROR IN 190
R:  READY.
```

This shows that you cannot directly write two numeric variables for each PRINT# and read them back in with INPUT#. Later, when we cover GET#, you will find out what PRINT# actually put on the cassette. Save this data cassette. Mark it BAD DATA.

b. Writing strings:

Character strings can be written to cassette with PRINT# just as they can be displayed on the TV with the PRINT statement.

Exercise: Modify SINGLE to work with a string variable. Reload SINGLE if necessary.

```
T:  120 PRINT#1,CHR$(64+I)
T:  190 INPUT#1,X$
T:  200 PRINT X$
```

This change should write the letters A to E on the cassette, one letter per line.

```
T:  RUN

R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY
T:  A
R:  OK
R:  A
R:  B
R:  C
R:  D
R:  E
R:  DONE
R:  READY
```

Exercise: Modify SINGLE to try two string variables per PRINT#.

```
T:  120 PRINT#1,CHR$(64+I),CHR$(64+I*I)
T:  190 INPUT#1,X$,Y$
T:  200 PRINT X$,Y$
T:  RUN
R:  PRESS PLAY & RECORD ON TAPE#1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  A     A     B     D
R:  C     I     D     P
R:  E     Y     1
```

A system crash will happen, i.e., keyboard does not respond. This is caused by reading (with INPUT#) beyond the EOT! The only way out is to power off and reload SINGLE.

Exercise: Fix SINGLE so that two strings can be written in each PRINT# and successfully read back with INPUT#. Reload SINGLE.

```
T:  120 PRINT#1,CHR$(64+I);",";CHR$(64+I*I)
T:  190 INPUT#1,X$,Y$
T:  200 PRINT X$,Y$
```

This inserts a comma between the characters PRINT#1 puts on tape.

```
T:  RUN
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  A     A
R:  B     D
R:  C     I
R:  D     P
R:  E     Y
R:  DONE
R:  READY.
```

Using what we learned here we can now go back and write out two numeric values and successfully read them back.

Exercise: Modify the SINGLE program to work with two variables for each PRINT#.

```
T:  120 PRINT#1,STR$(I);",";STR$(I*I)
T:  190 INPUT#1,X,Y
T:  200 PRINT X,Y
```

This change causes the information written to be the numeric character string of the first variable, followed by a comma (,), and then the numeric character string of the second variable.

```
T:  RUN
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  REWIND TAPE--THEN PRESS ANY KEY

T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK

R:  1      1
R:  2      4
R:  3      9
R:  4      16
R:  5      25
R:  DONE
R:  READY.
```

c. Testing Cassette I/O:

Some early PETs could not reliably read data files written with the PRINT #1 statement. To determine if your PET has this problem or not, try the tape reliability test in the following exercise. Be sure to use a high quality cassette. It is tempting to save a few dollars by buying low cost tape. DON'T DO IT unless you are willing to pay the cost in frustration. Use the best tapes that you can obtain. They will reduce, but not eliminate, your tape problems.

Exercise: Test the tape reliability on your PET. Doen!

```
T:  90 UN=1:NR=5000:EC=0
T:  95 OPEN UN,UN,2,"TEST DATA"
```

Set unit to test, number of test records and initialize error count. NR should be a large number. It takes about four minutes for each 1000 records. Be sure that your cassette is long enough for the NR that you chose.

```
T:  100 FOR I=1 TO NR
T:  110 PRINT#UN,I
T:  120 PRINT I
T:  130 NEXT I
T:  140 CLOSE UN
```

Write the numbers from 1 to NR one number at a time.

```
T:  150 PRINT "REWIND";UN
T:  160 PRINT "TYPE ANY KEY WHEN READY"
T:  170 GET A$:IFA$=""THEN 170
```

When finished rewind the tape and wait for that to be completed.

```
T:  180 OPEN UN,UN,0,"TEST DATA"
T:  190 FOR I=1 TO NR
T:  200 INPUT #UN,J
T:  210 IF I=J THEN 250
```

Read the numbers written back in and check to see if you get what you expect to.

```
T:  220 PRINT"ERROR-EXPECTED";I;"FOUND";J
T:  230 EC=EC+1
```

When an error is found, display the wrong number with the expected number. Then increment the error count.

```
T:  250 PRINT I
T:  260 NEXT I
T:  270 PRINT "FINISHED";EC;"ERRORS"
```

After reading NR data items, give error summary and stop.

```
T:  SAVE "TAPE TESTER",1,2
```

VERIFY as usual.

```
T:  RUN
```

The value of the data item being written to tape will be displayed. When the write portion has finished, a message will be displayed to rewind the tape so that the read check can be done. Follow the instructions in the messages. If the read check is successful, the message

FINISHED 0 ERRORS

will be displayed. If an error is found, the message

ERROR-EXPECTED n FOUND m

will be displayed. The "n" and "m" will be the actual expected and found numbers.

If the read check is successful, you may have a reliable tape writing system. But, don't get over confident. The problem seems to be a timing error. To be surer about your cassette, run TAPETESTER with a large value for NR on the kind of tapes that you plan to use for data storage. Also run the read check portion of TAPETESTER several times.

Exercise: Run only the read check portion of TAPETESTER.

To read check a tape that you wrote earlier, load TAPETESTER from tape.

```
T:  CLR
T:  UN=    :NR=    :EC=0:GO TO 150
```

Set UN to the unit that you want to use to check the tape on and NR to the value used when you wrote the tape. Reset the error count and start in the middle of TAPETESTER. You will get the same indications of success or failure described earlier.

If you are satisfied with the reliability of your PET, cassette unit, and tapes, you can skip the section (Section d.) on programming around the cassette problems. Before skipping, enter the following (simple) tape writing subroutine.

```
T:  NEW
T:  61000 PRINT #1,A6$
T:  61005 RETURN
```

Now proceed to Section e.

d. Programming around cassette problems:

To use cassettes for data storage it is necessary to write substantially more than the five or ten items that we have used in the earlier exercises. When a large volume of data is written, many blocks on tape are needed. The early PET cassette software has a bug which writes these blocks too closely together. When these blocks are read back in, errors usually occur. Turning the cassette motor on for a short period of time (about .1 second) between blocks improves read reliability.

Exercise: Prepare a subroutine to reliably write to cassette.

```
T:  NEW
T:  61000 REM WRITE RELIABLY TO TAPE #1
T:  61001 REM A6$ CONTAINS THE LINE TO WRITE
T:  61005 T6=TI:IF LEN(A6$)<190 THEN 61040
T:  61010 PRINT#1,LEFT$(A6$,189);
```

Special handling is required for very long lines. If the line to write is longer than the buffer, it is possible to write 2 blocks to tape in one PRINT#1 statement. To avoid that we split up long lines.

```
T:  61020 IF TI-T6> 120 THEN GOSUB 61900
```

When the PRINT#1 statement takes longer than 2 seconds, a tape write must have occurred. When it takes less time, the data was only entered into the cassette buffer.

```
T:  61030 T6=TI:PRINT#1,RIGHT$(A6$,LEN(A6$)-189)
T:  61035 GO TO 61050
```

Reset the timer and write the rest of the long line out.

```
T:  61040 PRINT#1,A6$
T:  61050 IF TI-T6 > 120 THEN GO SUB 61900
T:  61060 RETURN
```

Write the short line out and check the timer. This time test also checks the time it took to write the last part of the very long line. Use the subroutine to turn on the cassette motor if the timer shows that a cassette write occurred.

```
T:  61900 REM TURN ON THE CASSETTE MOTOR FOR .1 SEC.
T:  61905 POKE 59411,53:T6=TI
T:  61910 IF TI-T6 < 6 THEN 61910
T:  61920 POKE 59411,61:RETURN
```

The subroutine moves the cassette tape about .2 of an inch to give room between data records.

e. Keyboard wait subroutine:

```
T:  62990 PRINT "TYPE ANY KEY WHEN READY"
T:  63000 REM WAIT FOR KEY
T:  63010 GET A6$:IF A6$=""THEN 63010
T:  63020 RETURN
```

Wait for keyboard signal to continue.

Now save these routines on a separate cassette. We will use them as a starting point and add other useful routines later.

```
T:  SAVE "TAPELIB1",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING TAPELIB1
R:  READY.
```

Rewind cassette #1.

```
T:  VERIFY "TAPELIB1"
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  SEARCHING FOR TAPELIB1
R:  FOUND TAPELIB1
R:  VERIFYING
R:  OK
R:  READY.
```

We will add the main program to generate data for our write subroutine.

```
T:  100 PRINT "LOAD DATA TAPE"
T:  102 GO SUB 62990
T:  105 OPEN 1,1,2,"NEWDATA"
T:  110 FOR I=1 TO 200
T:  120 FOR J=1 TO 25
T:  130 A6$=STR$(I*J)
```

This creates 5000 data items to check. If you want more or less data to check, change the loop limits.

```
T:  140 GO SUB 61000
T:  150 NEXT J
T:  160 PRINT "RECORD"; I; "FINISHED"
T:  170 NEXT I
T:  180 CLOSE 1
T:  190 PRINT "WRITE DONE"
T:  200 STOP
```

Save on cassette the main program that generates data to test our write subroutine. Use a different tape than you used earlier for TAPELIB1.

```
T:  SAVE "TAPEWRITE",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING TAPEWRITE
R:  READY.
```

Since we have already gone through the verify sequence, we won't repeat it again. However, it is good practice to check files you have written. Now run the TAPEWRITE program. Allow yourself plenty of time. It should take about 20 minutes to write the 5000 data items. Make sure that your cassette is long enough for the amount of data that you intend to write.

```
T:  RUN
R:  LOAD DATA TAPE
R:  TYPE ANY KEY WHEN READY
T:  A
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  RECORD 1 FINISHED
    .
    .
    .
R:  RECORD 200 FINISHED
R:  WRITE DONE
R:  READY.
```

Rewind the cassette, remove it, mark it NEWDATA and save it for later.

2. INPUT#. The input statement (INPUT#) specifies the logical file number followed by a comma and the variable list.

   Example: INPUT#1,A

   The logical file number must match a logical file number in a previous OPEN with the input option specified.

   INPUT# reads from cassette the same as if you type keys at the keyboard and used an INPUT statement.

   a. Inputting numbers:

      Just as multiple data items from the keyboard must be separated by commas, so do multiple data items from the cassette.

   b. Inputting strings:

      Character strings can be read from cassette with INPUT# just as they can be read from the keyboard with the INPUT statement. That means that commas and some other special characters act as separators unless they are enclosed in quotation marks.

   c. Inputting and checking the test file:

      The cassette written in the PRINT# section (NEWDATA) will be read and checked here.

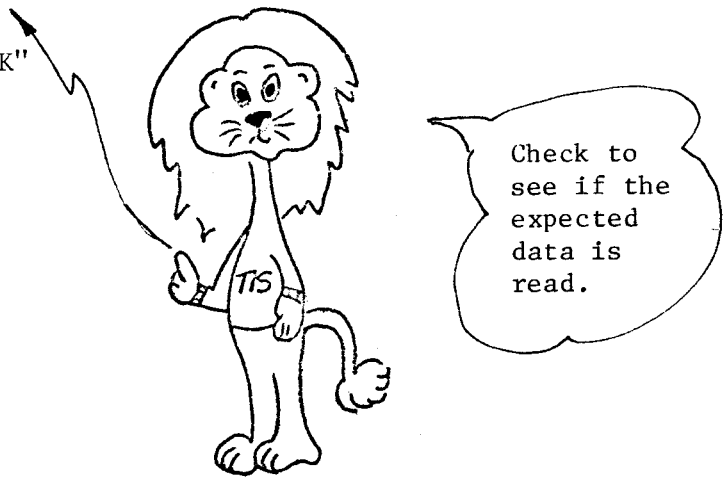
      Exercise: Read and check NEWDATA. First load TAPELIB1 so that we can use one of the subroutines.

```
T:  NEW
T:  LOAD "TAPELIB1"

R:              Usual load messages will appear here.

T:  100 PRINT "LOAD NEWDATA"
T:  110 GO SUB 62990
T:  115 OPEN 1,1,0, "NEWDATA"
T:  120 FOR I=1 TO 200
T:  130 FOR J=1 TO 25
T:  140 INPUT#1,X
```

145. Print I,J,X.

Read 5000 data items. If you wrote a different amount, you need to make the corresponding change here.

```
T:  150 IF X<>I*J THEN 300
T:  160 NEXT J
T:  170 PRINT "RECORD";I;"OK"
T:  180 NEXT I
T:  190 PRINT "DONE OK"
T:  200 STOP
T:  300 PRINT "ERROR";I;J
T:  310 STOP
T:  320 GO TO 160
T:  RUN
R:  LOAD NEWDATA
R:  TYPE ANY KEY WHEN READY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK
R:  RECORD 1 OK
    .
    .
    .
R:  RECORD 200 OK
R:  DONE OK
R:  READY.
```




Successful completion of this program verifies that the 5000 values were written and read back correctly.

3. GET#. The input statement (GET#) specifies the logical file number followed by a comma and a single variable.

   Example: GET#1,X$

The logical file number must match a logical file number in an OPEN with the input option specified.

GET# reads from cassette the same as if you typed keys at the keyboard and used a GET statement, with one exception. The exception is that data is always there except when an EOF or EOT is encountered. Using GET# allows you to check status and take recovery action.

a. Getting a number:

The only legal characters on the cassette for GET# are 0-9. Plus and minus are legal but you cannot distinguish between them and 0.

b. Getting a character:

Any character can be read from a cassette using GET#1,A$

c. Getting and checking the numeric file with two variables:

You will recall that in our experimentation with PRINT# we created a file with two variables for each PRINT#. We were unable to read the information back in with INPUT# and marked the cassette BADDATA. The GET# statement works differently than INPUT#. We will develop a program to display what is on the BADDATA tape.

Exercise: Display the contents of a data tape. Load TAPELIB1.

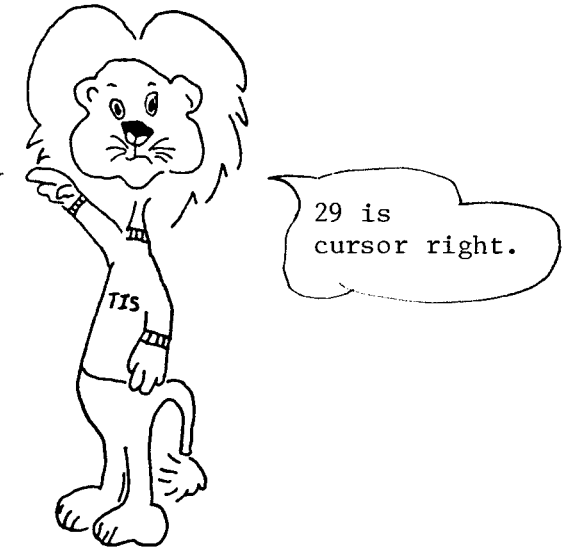
```
T:  100 PRINT "LOAD BADDATA"
T:  110 GO SUB 62990
T:  115 OPEN 1,1,0, "TEST"
T:  120 FOR I=1 TO 5
T:  125 B$="":C$=""
T:  130 GET#1,A$
T:  140 IFA$=CHR$(13) THEN 200
T:  150 B$=B$+"bb"+A$
```

Build a line with the character read and two leading blanks for spacing.

```
T:  160 X$=STR$(ASC(A$))
T:  165 IF LEN(X$)< 3 THEN X$=" b"+X$:GO TO 165
T:  168 C$=C$+X$
T:  170 GO TO 130
T:  200 PRINT B$
T:  210 PRINT C$
```

Build a second line with the numerical value of the character. Stretch the number to fill 3 columns, if it is less than three columns wide. After a carriage return is found, display the 2 lines.

```
T:  220 NEXT I
T:  230 STOP
T:  RUN

R:  LOAD BADDATA
R:  TYPE ANY KEY WHEN READY
T:  A
R:  PRESS PLAY ON TAPE #1
R:  OK

R:       1     1
R:   32 49 32 49 29
R:   32 50 32 52 29
R:
    .
    .
    .
R:  BREAK IN 230
R:  READY.
```




The character whose numeric value is 29 is the cursor right character. Evidently, INPUT# can not handle a cursor right.

d. Tape dump program:

You can use the program in the previous exercise to find out what is on any data cassette. The next exercise changes the program to make it more general.

Exercise: Generalize the tape dump program.

```
T:  100 INPUT"ENTER FILE NAME";FN$
T:  105 PRINT"LOAD";FN$
T:  115 OPEN 1,1,0,FN$
```

Request the name of the file to dump.

```
T:  224 GOSUB 62990
T:  226 GO TO 120
```

Display five lines, then wait for a key stroke before continuing with the next five lines.

Now save this program. Call it TAPESEE.

C. CLOSE

The CLOSE statement tells the system that you are finished with the logical file number.

1. Parameters. There is one required parameter. It is the logical file number. The form of the statement is CLOSE A. The logical file number can be a constant or a numeric variable whose value is between 0 and 255. Remember that the only legal file numbers for an OPEN are 1 to 255. Then the only file numbers we should use for CLOSE are 1 to 255.

2. End Processing. The type of end processing that is done when the CLOSE statement is executed is actually specified in the OPEN statement. In the OPEN statement you can select either an end of file (EOF) only or an end of file (EOF) and an end of tape (EOT). EOF only is option 1 and EOF followed by EOT is option 2. See Section II.A.1.c. for more information.

   a. End of file:

      When a CLOSE is done on a file that was OPENed with the end of file option specified, the PET writes two blocks. The last buffer is written followed by an EOF.

   b. End of tape:

      When a CLOSE is done on a file that was OPENed with the end of tape option specified, the PET writes the last buffer, a special EOF, and a special end of tape record.

## III. STATUS CHECKING

After each input or output statement, the system variable ST is updated to show the results of the last operation. Multiple errors can occur. When more than one error occurs, the individual error numbers are added together. ST is the sum of the error numbers.

There are two different types of records that can be written to cassette. Programs are written to cassette with the SAVE command. Data is written to cassette with the PRINT# statement. Each type has a different record (block) size. The size of the record written by the SAVE command depends on the size of the program you are saving. Records written with the PRINT# statement are first put in the 191 byte cassette buffer. Therefore each data record (block) is 191 bytes long.

### A. Errors

There are four different error conditions that are possible.

1. Short Block. The status of four (4) signals that a short block was read. A short record (block) was encountered before the expected number of bytes has been read. One cause is trying to read a short program (SAVEd) file instead of data.

2. Long Block. The status of eight (8) signals that a long block was read. The end of block signal was not encountered after the expected number of bytes had been read from that record (block). One cause is trying to read a long program (SAVEd) file instead of data.

3. Unrecoverable Read Error. The status of 16 signals that an unrecoverable read error was encountered.

4. Checksum Error. The status of 32 signals that a checksum error was found. A checksum is calculated over the data in memory and compared to a byte on tape. The checksum byte on tape was calculated before the tape was written. One cause is faulty memory (RAM) or multiple errors in the data on tape.

### B. End of File

The status of 64 signals that an EOF was found. This is not strictly an error. The status tells you where you are.

### C. End of Tape

The status of 128 signals that an EOT record was found. This can be used to tell you that you are finished with the data tape.

## IV. CASSETTE APPLICATIONS

Now that all of the statements for manipulating cassette files have been covered, we can put together a few programs using what we have learned.

### A. Keyboard to Cassette

To build a cassette data file we need some information to put in the file. A logical place to start is the keyboard.

1. Tape Support Subroutines. We will first develop some subroutines to add to TAPELIB1 which we used earlier.

   Exercise: Add subroutines to TAPELIB1 to make the rest of the cassette projects easier.

```
T: LOAD "TAPELIB1"
T: 62000 REM GET FILENAME-A6$ HAS TYPE
T: 62001 REM T6$ RETURNS THE FILENAME
T: 62010 PRINT "ENTER "; A6$;" FILE NAME"
T: 62020 INPUT T6$
T: 62030 IF LEN(T6$) >13 OR LEN(T6$) < 1 THEN 62010
T: 62040 T6$=T6$+".00":RETURN
```

Prompt for the file name. Make sure it is reasonable size. Append a sequence number to the file name and return.

```
T: 63100 REM UPDATE SEQUENCE NUMBER ON FILE NAME
T: 63110 T6$=STR$(VAL(RIGHT$(A6$,2))+1)
T: 63120 A6$=LEFT$(A6$,LEN(A6$)-2)+RIGHT$(T6$,2):RETURN
```

Take the last two characters of the file name, convert them to a number, add one and convert them back to a character string. Select all but the last two characters from the file name and put two digits from the updated sequence number on as the last two characters.

2. Upper and Lower-case Letters. Make a subroutine which treats a character typed at the keyboard as a normal typewriter. When shift and a letter are typed, make it upper-case. When just a letter is typed, make it lower-case.

```
T:  63200 REM HANDLE SHIFT AS TYPEWRITER
T:  63204 REM A6$ HAS CHARACTER TO CONVERT
T:  63210 B6%=0:T6=ASC(A6$)
T:  63215 IF T6> 64 AND T6< 91 THEN B6%=128
T:  63220 IF T6> 192 AND T6< 219 THEN B6%=-128
T:  63230 A6$=CHR$(T6+B6%):RETURN
```

This subroutine converts the character to its numeric equivalent. If it is a letter, we change it to swap cases. If it is not a letter, we don't change anything.

That completes the subroutines needed for keyboard to cassette.

```
T:  SAVE "TAPELIB2",1,2
R:  PRESS PLAY & RECORD ON TAPE #1
R:  OK
R:  WRITING
R:  READY.
```

Rewind the cassette and verify that the write was successful.

3. Main Program. The main program for keyboard to cassette will be developed next.

   Exercise: Build a main program to take data from the keyboard and write it on cassette. If you have turned power off or typed NEW since the last exercise, load TAPELIB2 first.

```
T:  100 DIMLI$(100)
T:  110 OP=1:CR$=CHR$(13)
```

Provide space for lines typed in. Initialize OP and CR$. OP is the EOF option for the file OPEN statement. CR$ is the carriage return character.

```
T:  120 LM=10:SAFE=300
```

Initialize LM and SAFE. LM is the maximum number of lines in a block to write to tape. You can change the limit to suit the application. The maximum is 100 unless you also change the DIM statement in line 100. The program prints a warning message when fewer than SAFE bytes of memory are left.

```
T:  130 A6$="WRITE":GOSUB 62000:WF$=T6$
T:  140 PRINT"LOAD ";WF$:GOSUB 62990
T:  150 PRINT"WHAT IS THE FINISHED SIGNAL?"
T:  160 INPUT EO$:EO$=EO$+CR$
```

Ask for the name of the file to write on. The FINISHED signal tells the program when you have entered all of the data.

```
T:  170 PRINT"GRAPHIC OR LOWER CASE?"
T:  180 INPUT A$:CASE$="GRAPHICS":POKE 59468,12
T:  190 IF A$="LOWER CASE"THEN CASE$="LOWER"
T:  200 IF A$="LOWER"THEN CASE$="LOWER"
T:  210 IF CASE$="LOWER"THEN POKE 59468,14
```

You have the option of using either the graphics symbols or the lower-case letters from the keyboard. The POKE 59468,14 statement will give you the lower-case letters.

```
T:  220 LP=0
T:  230 PRINT"START ENTERING DATA"
T:  240 A$=""
```

Initialize the line counter and the line buffer.

```
T:  250 GO SUB 63000
T:  260 IF CASE$="LOWER" THEN GOSUB 63200
T:  270 PRINT A6$;
T:  280 A$=A$+A6$:IF A6$<>CR$ THEN 250
```

This section gets a character from the keyboard, converts the case, if necessary, displays the character on the TV, and adds it to the line. This sequence repeats until the 'RETURN' key is typed.

```
T:  290 IF A$ <> EO$ THEN 500
T:  300 OP=2
T:  310 GO TO 600
```

Check to see if we are finished with all keyboard entries. If we are done, write the end of tape.

```
T:  500 LP=LP+1
T:  510 LI$(LP)=A$
T:  520 IF FRE(0)<SAFE THEN PRINT"WARNING SPACE LOW"
T:  530 IF LP<LM THEN 240
```

Update the line pointer, save the line just typed, and display a warning message if space is getting short. If there is room for more, go back for another line.

```
T:  600 PRINT"WRITING ";WF$
T:  610 OPEN 1,1,OP,WF$
T:  620 IF LP=0 THEN 700
T:  630 FOR I=1 TO LP
T:  640 A6$=LI$(I)
T:  650 GO SUB 61000
T:  660 NEXT I
```

Write a block of data on the cassette.

```
T:  700 A6$=WF$:GOSUB 63100:WF$=A6$
T:  710 CLOSE 1
```

Update the file name and close out the previous block.

```
T:  720 IF OP=1 THEN 220
T:  730 POKE 59468,12
T:  740 OPEN 1,1,2,WF$
T:  750 CLOSE 1
```

If there are more blocks, go back to read the keyboard. Otherwise, make sure we are in the standard (graphics) character set and stop.

```
T:  SAVE"KB2CASS",1,2
```

Save the keyboard to cassette program and verify it as usual.

B. Tape Copy

Now that you can enter data from the keyboard and write it on cassette, you may want to make copies of that information. Reading from one cassette and writing the information on another is a rather straightforward operation, if you have two cassettes. Since the basic PET has only one, we will set up the tape copy based on a single cassette unit.

1. Cassette Format. The format of the data on cassette is critical for the successful operation of the tape copy program. Since only a single cassette drive and buffer is available for both the read and the write cassettes, it is necessary to swap cassettes at the right time to complete the copy.

   To switch from read to write or back again, it is necessary to empty the cassette buffer. The only PET BASIC statement to empty a buffer is CLOSE. On a read file, CLOSE simply empties the buffer. On a write file, CLOSE writes (empties) the last buffer, and either writes an EOF or an EOF and an EOT. Once a file is CLOSEd, it is necessary to OPEN it before you can use it again. OPEN always searches for a file name record on a read file and always writes a file name (header) record on a write file.

   Because of the way OPEN and CLOSE operate, the only place to switch cassettes is after the EOF written by CLOSE and before the file name header written by OPEN. Fig. 4 shows the cassette format. We can only switch after an EOF; therefore, all data between EOFs must fit into memory at one time.

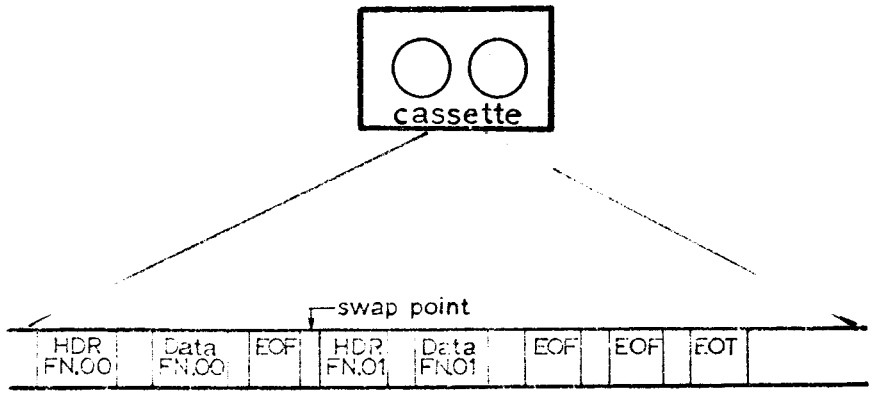



Fig. 4. Cassette Format

Since OPEN always writes a file name header (label) or searches for one, we need to adopt some labeling conventions. We could make all labels, except perhaps the first one, the null ("") label. This would simplify the program. However, it would make access to a part of a cassette very difficult. It would also make recovery from a cassette error difficult, if not impossible.

The labeling system that we will use is based on the file name given the cassette and a two digit "record" number. The first label (record) of a file is filename.00. Each successive record has the record portion of the label incremented by 1. So the labels are filename.00, filename.01, filename.02,. . .filename.99, filename.00. Only 100 unique labels are provided with this system. It is unlikely that more are needed. The process of loading, reading, writing, and unloading a cassette would have to be repeated 100 times to exhaust the unique labels. You probably wouldn't have the patience for that or the confidence that so many cassette input output operations could be done without human or machine error. Another fact to consider is that 100 records each containing 1500 bytes of data (on an 8K PET) is a total of 150000 bytes. If you have files that big, you probably want at least a second cassette or a different mass storage device.

2. Additional Subroutines. We only need a few more subroutines and a modification to one of our earlier subroutines to have the full set we need.

   Exercise: Modify the original enter filename subroutine to allow the user to specify a starting record number. Load TAPELIB2.

```
T:  62034 IF LEN(T6$)<3 THEN 62040
T:  62038 IF MID$(T6$,LEN(T6$)-2,1)="." THEN RETURN
```

When the file name entered has a period (.) in the proper place, we don't start the record number at 00.

Exercise: Add routines to prompt for unload and load of tapes. After each prompt wait for a response from the keyboard.

```
T:  62100 REM LOAD FILE MESSAGE
T:  62101 REM A6$ HAS THE NAME OF THE FILE TO LOAD
T:  62110 PRINT "LOAD FILE";A6$:GO TO 62990
T:  62200 REM UNLOAD FILE MESSAGE
T:  62210 PRINT "UNLOAD FILE ";A6$:GO TO 62990
```

Display the proper message and wait for a key to be typed.

This completes the subroutines needed for the cassette copy program. Save all the input-output subroutines.

```
T:  SAVE"TAPELIB3",1,2
```

VERIFY the copy when SAVE is complete.

3. Main Program. The main program for cassette copy uses the subroutines we have developed. The first step is to load these subroutines. The program is described in detail as you enter it. If you prefer an uninterrupted listing, the same program is in the APPENDIX.

   Exercise: A program to copy from one cassette to another.

```
T:  100 DIM LI$(100)
T:  110 OP=1:CR$=CHR$(13)
```

LI$ provides space for lines to be copies. OP is the EOF option for the file OPEN statement. CR$ is the carriage return character.

```
T:  115 ET=128:EF=64
T:  120 LM=20:SAFE=300
```

ET and EF are the status values for end of tape and end of file, respectively. The maximum number of lines (LM) can be changed. If you make LM greater than 100, you must change the DIM LI$ line. Print a warning message when you are close (within SAFE) to running out of space.

```
T:  124 A6$="READ":GO SUB 62000:RF$=T6$
T:  130 A6$="WRITE":GO SUB 62000:WF$=T6$
```

Get the read and the write cassette names.

```
T:  200 A6$=RF$:GO SUB 62100
T:  210 OPEN 1,1,0,RF$
```

Give mounting instructions for the read file and open it.

```
T:  220 LP=0
T:  300 INPUT#1,A$
T:  310 IF(ST=ET)OR ST=EF THEN 400
T:  320 LP=LP+1
T:  330 LI$(LP)=A$
```

Read lines from the input cassette and save them in the LI$ array. When an EOF or EOT is found, go write the buffer (array) out.

```
T:  340 IF FRE(0)< SAFE THEN PRINT"WARNING SPACE LOW"
T:  350 IF LP<LM THEN 300
T:  360 PRINT"TOO MANY LINES IN FILE TO COPY"
T:  370 STOP
```

Give a warning or error message, if we run out of space.

```
T:  400 SS=(ST)AND ET
T:  410 CLOSE 1
```

Save the status information and close the input (read) file.

```
T:  420 A6$=RF$:GO SUB 62200
T:  430 A6$=RF$:GO SUB 63100:RF$=A6$
```

Prompt to unload read file. When it is unloaded, update the read record number.

```
T:  500 A6$=WF$:GO SUB 62100
T:  510 PRINT "WRITING ";WF$
T:  600 IF LP=0 OR SS=ET THEN OP=2
T:  610 OPEN 1,1,OP,WF$
```

Prompt for mounting of the output (write) cassette and then open it.

```
T:  620 IF LP=0 THEN 670
T:  630 FOR I=1 TO LP
T:  640 A6$=LI$(I)
T:  650 GO SUB 61000
T:  660 NEXT I
T:  670 LP=0:CLOSE 1
```

Write out all the lines in the buffer and close the write file.

```
T:  680 A6$=WF$:GO SUB 62200
T:  700 A6$=WF$:GO SUB 63100:WF$=A6$
```

Display the write cassette unload message and then update the output record number.

```
T:  710 IF OP<>2 THEN 200
T:  720 PRINT "DONE"
T:  730 STOP:CLR:GO TO 100
```

Repeat the copy cycle until an EOT is found on the input (read) cassette.

```
T:  SAVE "CASS2CASS",1,2
```

## V. CASSETTE PERFORMANCE

### A. Data Rate

A simple way to determine the data rate used by the recording scheme on your PET is to write a known amount of information on a tape and time how long it takes to read (or write) that information.

Exercise: Determine PET cassette data rate. Take a program cassette and load it. Write down the time for each message.

| | | TIME |
|---|---|---|
| T: | LOAD "CASS2CASS" | -- |
| R: | PRESS PLAY ON TAPE #1 | :30 |
| R: | OK | :30 |
| R: | SEARCHING FOR CASS2CASS | :30 |
| R: | FOUND CASS2CASS | :44 |
| R: | LOADING | :44 |
| R: | READY. | 1:28 |
| T: | ?FRE(0) | |
| R: | 5481 | |
| R: | READY. | |
| T: | NEW | |
| T: | ?FRE(0) | |
| R: | 7164 | |

The name label (header) is about 14 seconds from the beginning of tape. The two FRE prints show that this program is 7164-5481=1683 bytes long. It took 88-44=44 seconds for a data rate of about 1683/44 = 38 bytes per second.

### B. Reliability

Early PETs have some difficulty in reading cassettes. There are a few steps that you can take to improve the situation.

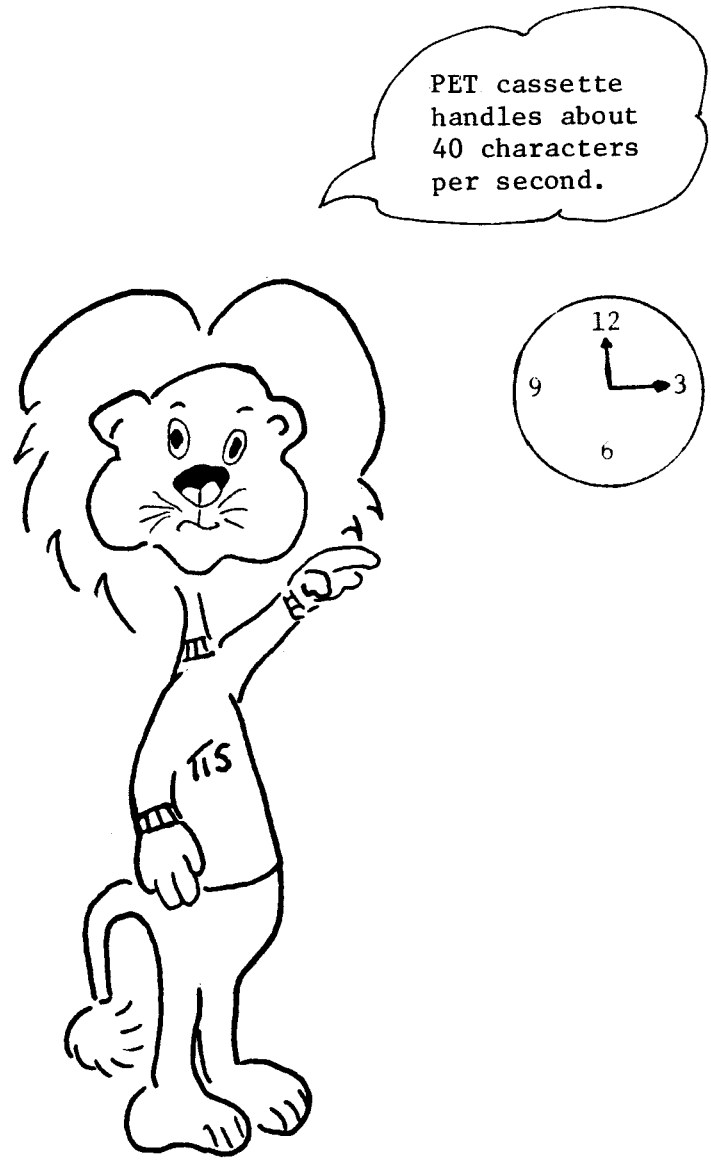
1. Always wait for the PET to prompt you to press buttons on the cassette.

2. Use a routine to turn on the cassette between data blocks.

3. Keep the tape heads cleaned and degaussed.

4. When you have two programs on a cassette and you have difficulty loading the second, load the first (with LOAD) and then load the second with (LOAD). If you can recover the second, try

changing the length of the first (one byte shorter sometimes works miracles). Or put the second program on a separate cassette.



## VI. APPENDIX

Program Listings: Complete sequential listing of TAPELIB3 the subroutines to make cassette input-output easier.

```
61000 REM WRITE RELIABLY TO TAPE #1
61001 REM A6$ CONTAINS THE LINE TO WRITE
61005 T6=TI:IF LEN(A6$)<190 THEN 61040
61010 PRINT#1,LEFT$(A6$,189);
61020 IF TI-T6>120 THEN GO SUB 61900
61030 T6=TI:PRINT#1,RIGHT$(A6$,LEN(A6$)-189)
61035 GO TO 61050
61040 PRINT#1,A6$
61050 IF TI-T6>120 THEN GO SUB 61900
61060 RETURN

61900 REM TURN ON THE CASSETTE MOTOR FOR .1 SEC
61905 POKE 59411,53:T6=TI
61910 IF TI-T6<6 THEN 61910
61920 POKE 59411,61:RETURN

62000 REM GET FILENAME-A6$ HAS TYPE
62001 REM T6$ RETURNS THE FILENAME
62010 PRINT"ENTER";A6$;"FILE NAME"
62020 INPUT T6$
62030 IF LEN(T6$)>13 OR LEN(T6$)<1 THEN 62010
62034 IF LEN(T6$)<3 THEN 62040
62037 IF MID$(T6$,LEN(T6$)-2,1)="."THEN RETURN
62040 T6$=T6$+".00":RETURN

62100 REM LOAD FILE MESSAGE
62101 REM A6$ HAS THE NAME OF THE FILE TO LOAD
62110 PRINT "LOAD FILE";A6$:GO TO 62990
62200 REM UNLOAD FILE MESSAGE
62210 PRINT "UNLOAD FILE";A6$:GO TO 62990

62990 PRINT "TYPE ANY KEY WHEN READY"
63000 REM WAIT FOR KEY
63010 GET A6$:IF A6$=""THEN 63010
63020 RETURN

63100 REM UPDATE SEQUENCE NUMBER ON FILENAME
63101 REM A6$ CONTAINS THE FILENAME
63110 T6$=STR$(VAL(RIGHT$(A6$,2))+1)
63120 A6$=LEFT$(A6$,LEN(A6$)-2)+RIGHT$(T6$,2):RETURN

63200 REM HANDLE SHIFT AS TYPEWRITER
63204 REM A6$ HAS CHARACTER TO CONVERT
63210 B6%=0:T6=ASC(A6$)
63215 IF T6>64 AND T6<91 THEN B6%=128
63220 IF T6>192 AND T6<219 THEN B6%=-128
63230 A6$=CHR$(T6+B6%):RETURN
```

Program Listings: Complete sequential listing of KB2CASS, the program to write data typed on keyboard to the cassette. Load TAPELIB3 first!

```
100 DIM LI$(100)
110 OP=1:CR$=CHR$(13)
120 LM=10:SAFE=300
130 A6$="WRITE ";GO SUB 62000:WF$=T6$
140 PRINT "LOAD ";WF$:GO SUB 62990
150 PRINT "WHAT IS FINISHED SIGNAL?"
160 INPUT EO$:EO$=EO$+CR$
170 PRINT "GRAPHIC OR LOWER CASE?"
180 INPUT A$:CASE$="GRAPHICS":POKE 59468,12
190 IF A$="LOWER CASE" THEN CASE$="LOWER"
200 IF A$="LOWER" THEN CASE$="LOWER"
210 IF CASE$="LOWER" THEN POKE 59468,14
220 LP=0
230 PRINT "START ENTERING DATA"
240 A$=""
250 GO SUB 63000
260 IF CASE$="LOWER" THEN GO SUB 63200
270 PRINT A6$;
280 A$=A$+A6$:IF A6$ <> CR$ THEN 250
290 IF A$ <> EO$ THEN 500
300 OP=2
310 GO TO 600
500 LP=LP+1
510 LI$(LP)=A$
520 IF FRE(0) < SAFE THEN PRINT"WARNING SPACE LOW"
530 IF LP < LM THEN 240
600 PRINT "WRITING ";WF$
610 OPEN 1,1,OP,WF$
620 IF LP=0 THEN 700
630 FORI=1 TO LP
640 A6$=LI$(I)
650 GO SUB 61000
660 NEXT I
700 A6$=WF$:GO SUB 63100:WF$=A6$
710 CLOSE 1
720 IF OP=1 THEN 220
730 POKE 59468,12
740 END
```

Program Listings: Complete sequential listing of CASS2CASS, the program to copy from one cassette to another. Load TAPELIB3 first!

```
100 DIM LI$(100)
110 OP=1:CR$=CHR$(13)
115 ET=128:EF=64
120 LM=20:SAFE=300
124 A6$="READ":GO SUB 62000:RF$=T6$
130 A6$="WRITE":GO SUB 62000:WF$=T6$
200 A6$=RF$:GO SUB 62100
210 OPEN 1,1,0,RF$
220 LP=0
300 INPUT#1,A$
310 IF(ST=ET) OR ST=EF THEN 400
320 LP=LP+1
330 LI$(LP)=A$
340 IF FRE(0)<SAFE THEN PRINT"WARNING SPACE LOW!"
350 IF LP< LM THEN 300
360 PRINT "TOO MANY LINES IN FILE TO COPY"
370 STOP
400 SS=(ST)AND ET
410 CLOSE 1
420 A6$=RF$:GO SUB 62200
430 A6$=RF$:GO SUB 63100:RF$=A6$
500 A6$=WF$:GO SUB 62100
510 PRINT "WRITING ";WF$
600 IF LP=0 OR SS=ET THEN OP=2
610 OPEN 1,1,OP,WF$
620 IF LP=0 THEN 670
630 FOR I=1 TO LP
640 A6$=LI$(I)
650 GO SUB 61000
660 NEXT I
670 LP=0:CLOSE 1
680 A6$=WF$:GO SUB 62200
700 A6$=WF$:GO SUB 63100:WF$=A6$
710 IF OP<>2 THEN 200
720 PRINT "DONE"
730 STOP:CLR:GO TO 100
```

# Notes

EoF - EoT 1-7.

ST 3-1

# Notes

# Notes