File-Handling Within FORTRAN

MALCOLM C. HARRISON
Courant Institute of Mathematical Sciences,
New York University, New York

This note describes some FORTRAN subroutines to facilitate handling of tape files. They allow symbolic naming of information files, without violating the casual scientific programmer's idea of simplicity. Some comments on two years use of these subroutines are given.

Introduction

In any large system of programs making extensive use of magnetic tape, it is convenient to use some consistent method for the allocation of permanent and temporary tape storage. In particular, a system of programs which communicate via magnetic tape, and which may be the work of many programmers, runs the risk of growing extremely ponderous if every reference to magnetic tape requires knowledge of the whole data structure. Also, even trivial alterations or improvements to such a system sometimes involve so much work that they are just not worth the trouble.

The subroutines described below constitute a particularly simple solution to this problem, and have been found very effective for the casual scientific programmer. The subroutines are written in FORTRAN II and FAP for the IBM 709/90/94, and allow complete freedom in the form in which the data is written on the tape, including the standard FORTRAN input/output statements.

Method

The unit of information is the file, which may consist of any number of records. Each file is given a symbolic name by the programmer when written on the tape, and subsequent references to this file are made by using this name.

On the tape each file is terminated by an end-of-file mark, and is preceded by an *identifying record* (IDR) containing the names of the file and the names of all the preceding files on that tape. This enables any file to be located by reading the record after any end-of-file mark, and searching it for the required file name. If it is found, the tape is rewound the required number of files. Otherwise, the tape is moved forward until the file is located.

All the tape manipulation is done by three subroutines: LOAD, FILE and FILEND. The algorithms used by the subroutines are as follows:

LOAD (NTAPE, TAPNAM)
1. Rewind NTAPE, and write EOF.
2. Write identifying record containing TAPNAM, 3 (the number of words in the record), and zero (denoting the last file).
3. Write EOF, and backspace over 2 EOFs.

The work presented in this paper was supported by the US Atomic Energy Commission under Contract no. AT(30-1)-1480.

FILE (FILNAM, NTAPE, TABLE)
1. Move NTAPE forward over first EOF.
2. Read identifying record tape, which will contain first the name of the tape, then the number of words in the record, and then the names of the files up to and including the next. Store in TABLE.
3. Search the file names for FILNAM.
4. If not found, and if not the last file on the tape (denoted by a zero in the last file-name position), move tape forward over EOF, and go to 2.
5. If not found, and if last file, replace zero with FILNAM, rewrite identifying record followed by EOF, backspace over EOF, and return.
6. If found, backspace tape over correct number of EOFs, read and check the identifying record after the EOF, and return. If the record is not correct, print an error message, and terminate the job.

FILEND (NTAPE, TABLE)
1. Write EOF on NTAPE.
2. Add zero to the list of file names in TABLE, to denote that there are no more files, and step up the word counter.
3. Write identifying record, followed by EOF.
4. Backspace over 2 EOFs.

The subroutine LOAD prepares a scratch tape to use the remaining subroutines FILE and FILEND.

The subroutine FILE (FILNAM, NTAPE, TABLE) searches the tape on unit NTAPE for the file named FILNAM. If it finds it, the tape is left positioned after the IDR ready to read the information. If it does not find the file, it assumes that the file is about to be written so it writes an IDR after the last file on the tape containing TAPNAM and the names of all the files together with FILNAM. It then writes an end-of-file mark, and leaves the tape positioned just after the new IDR. TABLE is an array of temporary storage in which FILE leaves the IDR it has written or read last, and which must be at least three words longer than the maximum number of files.

The subroutine FILEND (NTAPE, TABLE) is used to terminate a file which has been written on NTAPE. It writes an end-of-file mark, followed by an IDR containing a zero to denote that there are no files on the tape following, and a further end-of-file mark. The tape is then backspaced over the two end-of-file marks, and is again ready for use.

It should be noted that the name of the tape TAPNAM is included in each IDR, but never referred to. This is in anticipation of an improved version, where the tapes may also be given symbolic names. TAPNAM and FILNAM are normally written into the program as Hollerith characters. The example given below illustrates the use of the subroutines.

```
DIMENSION TABLE (10)
...
NTAPE = 11
...
CALL LOAD (NTAPE, 6HTAPE-Z)
...
CALL FILE (6HFILE-1, NTAPE, TABLE)
WRITE TAPE NTAPE, A, B, (C(I), I = 1, 10)
CALL FILEND (NTAPE, TABLE)
...
```

```
      CALL FILE (6HFILE-3, NTAPE, TABLE)
      WRITE OUTPUT TAPE NTAPE, 100, D, E
100   FORMAT 2A6
      WRITE TAPE NTAPE, F, G
      CALL FILEND (NTAPE, TABLE)
      ...
      CALL FILE (6HFILE-1, NTAPE, TABLE)
      READ TAPE NTAPE, AA, BB, (CC(I), I = 1, 10)
      ...
      CALL FILE (6HFILE-5, NTAPE, TABLE)
      WRITE TAPE NTAPE, I, J
      CALL FILEND (NTAPE, TABLE)
      ...
      CALL EXIT
      END
```

Comments

Several consequences of the algorithms may be noted. Rewriting a file has the effect of erasing any files which may have followed it. An attempt to read a nonexistent file will result in an end-of-file being read (and a consequent error exit in FORTRAN). No information is kept in core about the contents of the tapes (except between FILE-FILEND pairs when writing a file), so tapes may be removed, changed, rewound, or have their positions altered during a program, except when files are actually being used.

The subroutines described here have been in use for over two years, in programs as different as the calculation of molecular wave functions and automatic differentiation of formulas. One particular advantage of their use has been the ease with which programming effort has been split between many people, with each programmer being able to read or write files on any tape with the knowledge that the operation of other programs will not be affected. Also, programs using only a small portion of the data have been written with the knowledge of this data alone.

The present simple subroutines are, of course, only a first approximation to the solution of the tape-manipulation problem, but they have proved so successful that the author cannot foresee himself writing another program using tapes without some such procedure. In fact, he would suggest that the symbolic naming of information stored on tape should be considered for incorporation into the scientific programming languages.

Acknowledgments. The subroutines described here were written while the author was engaged in molecular calculations at the MIT Cooperative Computing Laboratory sponsored by the National Science Foundation, the Office of Naval Research, and the Air Force Office of Scientific Research.

RECEIVED FEBRUARY, 1965

REFERENCES

1. HARRISON, M. C. File-handling routines. C.C.L. Tech. Note 19, MIT, Cambridge.
2. CSIZMEDIA, I. G., HARRISON, M. C., MOSKOWITZ, J. W., SEUNG, S., SUTCLIFFE, B. T., AND BARNETT, M. P. The POLYATOM system, part I: basic subroutines. C.C.L. Tech. Note 36, MIT, Cambridge.

PRACNIQUES

The Techniques Department is interested in publishing short descriptions of Techniques which improve the logistics of information processing. To quote from the policy statement, Communications of the ACM 1 (Jan. 1958), 5: "It is preferable that techniques contributed be factual and in successful usage, rather than speculative or theoretical. One of the major criteria for acceptance and the question one should answer before submitting any material is—Can the reader use this tomorrow?" Clear, concise statments of fairly well-known but rarely documented methods will contribute significantly to raising the general level of professional competence.—C.L.McC.

NEGATIVE AND ZERO SUBSCRIPTS IN FORTRAN II PROGRAMMING FOR THE IBM 1620

The requirement that subscripts be unsigned integers creates some inconvenience in FORTRAN programming for summarization of completed questionnaires in which the responses may be scaled beginning at zero.

Suppose that a response range is the set of integers from 0 to 4. The array can be dimensioned as M(5), and tallied by repetitive reading of the response I, followed by the statement M(I+1) = M(I+1) + 1. If there is a large number of questionnaires with many categories execution time is increased because of the necessity for subscript modification.

The apparent inability to employ zero as a subscript seemed illogical under appropriate conditions. Also, circumvention of the published requirement that integer subscripts be unsigned and positive would be of advantage. The possibility of the extension of the range of subscript values was therefore investigated on the IBM 1620 (*without index registers*) both in FORMAT FORTRAN and in FORTRAN II. It was found that the requirement is imposed by the processor and not by virtue of programming logic. Thus, while

```
M(0) = M(0) + 1
```

will not compile, the satement I=0, followed by M(I) = M(I) + 1 will compile, and operate correctly if appropriate storage is reserved.

In FORMAT FORTRAN storage is reserved by serial listing in the dimension statement. For the problem above the specification is:

```
DIMENSION L(1), M(4)
```

In FORTRAN II the equivalence statement can be used to advantage:

```
      DIMENSION L(5), M(4)
      EQUIVALENCE (L(2), M)
```

The remainder of the routine for N values (N assumed available) is:

```
      DO 1 I = 1, 5
1     L(I) = 0
      DO 2 J = 1, N
      READ 10, I
2     M(I) = M(I) + 1
      J = 0
      PRINT 11, (I,M(I), I = J,4)
```

The routine was useful in tallying 142 one-way categories of approximately 600 questionnaires (140 categories were scaled beginning at zero).