INTERNAL REPORT #86

SRL Chapter/Verse Format

by

T. L. Garrard

N. S. Collins

Space Radiation Laboratory

California Institute of Technology

Pasadena, California

9/21/81
(revised 10/8/81)

SRL Chapter/Verse Format

1. Description

In order to maximize flexibility and minimize tape length, tapes should be formatted with lots of small logical blocks squeezed together into long physical records. The blocking technique described below was worked out for the HEAO C-3 data analysis but is clearly adaptable to other projects. It should be treated as a SRL standard and utilized whenever possible. A substantial library of programs exists already for handling data in this format and more are being written.

The blocking technique is reminiscent of IBM System 370 VB or VBS blocking, but is much more visible to the user, much less demanding on the system, and much less prone to catastrophic data loss to tape errors. Data is organized into logical blocks called chapters. Chapters are placed in a buffer as they are generated, and when chapter input threatens to overflow the buffer, the buffer is flushed to tape. This process is handled by a library routine called putchap. Normally only the data is written, not the trailing, empty portion of the buffer. Thus records are of variable length, but always contain an integral number of chapters. Each chapter begins with a 2-byte "key" integer which specifies the type of chapter. All chapters of this type have identical lengths. When reading tape, chapters are retrieved from the input buffer by a library routine called getchap. Getchap finds chapters using a table of chapter lengths which is indexed by the chapter key. This table is written on each tape in the first record on the tape, in a special chapter with key = 0. Chapter 0 has fixed a format, known to getchap.

An embellishment of the scheme allows chapters to be broken down into verses. Offsets of the verses within a chapter are also specified in Chapter 0.

The decision of what constitutes a "logical block", i.e., a chapter or verse is up to the user but some guidelines are clear. Since the key imposes a 2-byte overhead on each chapter, very short chapters are inefficient in their tape usage. Since a physical record length of ~4K to 8K bytes is appropriate to PDP-11 applications with 1600 bpi tapes, very long chapters do not fit into the output buffer efficiently. Thus chapters should be between ~20 bytes and ~400 bytes long. Any point in a logical format which seems a likely candidate for insertion of additional items at some future time, is a good point for a chapter or verse break. Example:

current format			new format		
variable	verse	word	variable	verse	word
r	1	0	r	1	0
theta	1	1	theta	1	1
X	2	0	phi	1	2
Y	2	1	X	2	0
Z	2	2	Y	2	1
			Z	2	2

Note that since X (& Y & Z) is in a separate verse it is still addressed as word 0 of verse 2 in either format. Thus no program changes are required in programs which read this data when the new format is introduced.

A list of chapters is maintained by Tom Garrard, and selection of chapter numbers should be done in consultation with him. Documentation for existing chapters is on /usr/tlg/dpgeneral/chap.dc. Note that some special chapters should be used by all users for consistency:

Chapter 0	specifies format
Chapter 101	specifies end-of-record
Chapter 102	specifies end-of-interval
Chapter 103	specifies end-of-tape
Chapter 105	specifies end-of-single-file
Chapter 206	specifies end-of-plot

In addition, a number of existing chapters can be used by other users. Chapter 1 specifies time for both HEAO C-3 and Voyager. Chapters 202-204 are Voyager display points, and can quite likely be used as they are by other projects.

2. Programming

Although the previous section applies only to tapes, the following routines work on both disk and tape for compatibility. Disk records must always be the same length (currently 2K bytes) so for disk applications, very large chapters will waste disk space.

To make a tape in chapter/verse format, this is the general idea:

1) Get a unique set of chapter numbers from Tom G.
2) Include (with a *#include* statement) <chap.h> in your program.
3) For each different chapter format, define a global array of 14 integers. i.e:

   int chpA[14] = { length, v1, v2, v3, ... , v12, numverses };

   where length is the total number of bytes in the chapter (including the 2-byte key), v1, etc. is the (byte) offset from the start of the chapter to the beginning of the verse. The first data byte in a chapter has an offset of 2. The last entry is the number of verses actually used. Unused verses still need to be given an offset (usually 0).

4) Inside the program, before *putchp()* is called the first time, for each chapter used, the internal chapter array needs to be initialized:

   chapter[*key*] = chpA;

   where chpA is the name of the corresponding array for chapter *key*. (The chapter array is declared in <chap.h>, and is otherwise unused in the user program).

5) Call *putchp(key)*. It returns a pointer to a space large enough to hold data for a chapter of type *key*.

6) Fill the buffer space by assigning and incrementing the pointer. See the C manual for the "++" operator and pointer usage. If the data is already in an array, see *movechp()* below.

7) Repeat 5) and 6) until done. The routines will take care of output names and whether the output medium is filled. No bookkeeping needs to be done in the program.

8) When done, call *putchp(C_EOT)*.

To read a tape in chapter/verse format:

1) Include <chap.h>
2) Call *getchp()*. It returns the key number of the next chapter.
3) Call *getvrs(N)*. It returns a pointer to verse *N*.
4) A key of *C_EOT* (103) means end of input.

When running programs using chapter/verse format, when a name is required, it prompts on the terminal for input or output file. If a tape is being used, enter the unit number. If a disk file is being used, enter the name of the file. Disk file names cannot begin with numbers. They can contain numbers, but the first character must be alphabetic.

When the end of an input file is reached, it asks for the next input. Thus it is easy to add files or tapes together. When the last input is reached, enter "-1" for input and the program will quit. When the end of an output tape is reached (disk files should never have this problem) it prompts for the next output tape, so both input and output can be continued over many different tapes and files.

3. Chapter Routines

The routines will someday be installed in the default library, so they will be automatically included with your program. For now, they are in their own library and are included by " -lchap " when compiling.

Predefined Values

C_EOF and *C_EOP* are defined as 103 and 206 respectivly, for end-of-file and end-of-plot.

getchp()

returns the key number of the next chapter on the input file.

*char *getvrs(N)*
: must be called after ***getchp()***. Returns a pointer to verse *N* of the current chapter (the one returned by the last ***getchp()***).

*char *putchp(key)*
: returns a pointer to an area large enough for a chapter of type ***key***.

copychp(key)
: assumes ***key*** to be the number of the last chapter read by ***getchp()***, and copies it to the output file. Can be used to copy a whole file like this:

```
copychp(getchp());
```

movechp(key,pointer)
: called instead of *putchp(key)*, it allocates space for a chapter ***key***, and copies the data starting at *pointer* into it.

rewchp()
: rewinds the input unit. If it's a disk file, it seeks to the beginning of the file.

4. Multiple Units

There are also routines for handling multiple input and output units. Generally speaking, they are the same as the above, with the letter "m" prepended and an extra argument for the unit number. Unit numbers can be 0 or 1. If there are two input tapes, any chapters on both tapes that are going to be used must have the same format.

mgetchp(U)
*char *mgetvrs(U,N)*
*char *mputchp(U,key)*
mrewchp(U)
: same as above, but *U* is the input or output unit number (0 or 1).

mcopychp(Ufrom,Uto,key)
: assumes the last chapter read on unit *Ufrom* was of type ***key***, and copies it onto unit *Uto*.

mmovechp(U,key,pointer)
: used instead of *mputchp(U,key)*, it writes a chapter of type ***key*** on unit *U* using the data starting at *pointer*.