Small Runoff

by Joseph Felsenstein and
Randyl Britten

Introduction

Small Runoff is a text formatting program written in a highly portable language called Small [1]. It is based directly on Seattle Runoff originally written by Felsenstein in UCSD Pascal and converted to Small by Britten. Both versions are essentially the same and are based, in turn, on the text formatting program described by Kernighan and Plauger in their book, Software Tools [2]. The particular mnemonics used for commands in [2] have been altered to make them more compatible with the many programs called "Runoff" which are available on PDP-11, DEC-10, CDC 6000, and other computers. As an example of its capabilities, this paper is produced by Small Runoff, and both the input and output files are available on CP/M compatible floppy disk.

Much of the code is directly from [2], with some differences in the way strings are handled and some new features added. In many cases, the syntax of the new features has been made as close as possible to the DEC family of Runoffs. String handling in both Pascal and Small avoids the use of an end-of-string marker technique used in [2]. The Pascal versions use string handling intrinsic procedures where available. Small represents a string as an array with the zero array element containing the length of the string. All string manipulation procedures needed are written in Small.

Felsenstein converted the UCSD Pascal version to Pascal MT+ to improve both its performance and its portability. This version of the program is a 15k .COM file that will run under CP/M in systems as small as 32K. It was compiled in Pascal MT+, version 5.2. The version in Small is a 17k .COM file that actually uses less space than the Pascal MT+ version because Pascal uses additional memory for stack and heap space. Small needs no heap space, and stack requirements are minimum since only return addresses are stacked.

Command Structure

Runoff commands instruct the program as to how to process lines of text. Both text and commands are in the same file. The commands are placed on separate lines in the text and are identified by a period in column one. For example, the command .C 4 instructs the program that the next four lines of text are to be centered, the command .RM 40 sets the right margin to 40 from that point on, the command .P 6 instructs the program to start a new paragraph, with an indenting of six spaces, and the command .J tells the program that all text from that point on is to be right justified.

To execute Runoff in a CP/M system, simply type "RUNOFF<cr>", where "<cr>" indicates the "Return" character. Runoff prompts the user for two file names, then reads lines from one file and writes to the other. The Small Runoff version allows CON: or TTY: for console output, and LST: for printer output file names.

If a command is not recognized, an error message will be written to the screen, and processing will continue without taking any special action on that command.

The input file consists of lines, each of which is either text or a command. Each command must be on its own line, at the beginning of the line. Each command consists of a period "." followed by one or more letters. Optionally the command line may also contain a parameter. If so, it is separated from the command letters by at least one blank. The parameter consists of one or more digits, optionally preceded by a plus "+" or minus "-" sign. The flag command ".FL c" is a special case that has a single character "c" as its parameter, separated of course from the first part of the command by a blank. The flag command sets the pseudo-blank character, described later, to "c".

The following are legal commands:

```
                   .C 3
                   .C 03
                   .C +4
                   .C -5
                   .CENTER    34
```

and the following are not legal commands:

```
                   .CEN TER 3      --will be processed but the 3 will be lost
                   .C 45  .BL 3    --only the first command will be found
                   .C 0.33         --number not an integer, .33 will be lost
```

There are default values for all parameters, and these apply until a value has been assigned. The presence of a + sign before the integer means that its value is to be INCREASED by that amount, the presence of a - sign means that it is to be DECREASED by that amount, and the absence of a sign means that it is to be SET to that amount. So .BL 6 and .BL +6 mean different things. Each parameter in the program has a maximum and minimum allowed value, and if the value is set to beyond that range, the minimum (or maximum) value will be used instead. So .IN 200 (indent 200 spaces) will set the indent value to the highest reasonable value, and causes indenting all the way to the right margin but not beyond.

Many of the commands cause a "break", that is, when they are encountered they cause the current output line which is being built up to be output without any filling or justification before the parameter is acted upon. Thus .C, which centers the following line, also causes output of the text preceding it before it is acted upon.

A TABLE OF COMMANDS

In the following command table the unique part of each command is capitalized, and the non-unique part is in lower-case. The program reads until it recognizes the command as unique, and then skips to the next blank to find the parameter value (if any). So since the command .SKIP is recognized as soon as the S and the K have been read, the I and the P are optional. The program shifts all commands to upper-case before looking at them, so any command letter may be in either upper or lower case.

COMMAND	causes a break?	default value	WHAT IT DOES
.BLank n	Y	1	issues n blank lines
.BM n	N	3	bottom margin is n lines
.BReak	Y		causes a break
.Center n	Y	1	centers the next n lines
.COmment	N		this line is a comment and is ignored
.Fill	Y	(on)	turns on filling
.FL c	N	#	from now on character c is the one which will be the pseudo-blank
.FOoter n	N	2	read next line in as the footer (the title at the page bottoms) to go on line n of bottom margin
.Header n	N	2	read next line in as the header (the title at the page tops) to go on line n of top margin
.Indent	Y	0	indent next line to column n
.Justify	Y	(on)	turns on justifying
.Lm n	Y	0	set left margin to column n
.NOFill	Y	see .F	turns off filling
.NOJustify	Y	see .J	turns off justifying
.NUmber	N	+1	sets the next page number without causing a page break
.P n	Y	5	start paragraph, indent n (SEE NOTE 1)
.PP n	Y	5	" " " " " " "
.PAge n	Y	+1	start a new page here (page number n)
.PG n	"	"	" " " " " " " "
.PL n	N	66	set page length to n
.Rm	Y	72	set right margin to column n
.SKip	Y	1	issues (n * sp) blank lines, where sp is the line spacing
.SPacing	N	1	set spacing to n (1 is single-spaced)
.TM n	N	3	top margin is n lines
.TP n	Y	0	if within n lines of end of page, start a new page at this point
.Ul n	N	1	underline next n input lines

TWO NOTES

Note 1: In the case of the paragraph command ".P n", the parameter value n is the amount (positive or negative) by which the indent in the first line of a paragraph exceeds the normal indent. That is, a .P +10 has the same effect as a .P 10, unlike the case in the other commands. The parameter value in this command will hold sway until the next numerical value is typed in, so that if .P 10 is followed some lines later by .P, the latter has the same effect as if it too said .P 10. Note that a negative indent may be useful in forming lists of items.

Note 2: When filling is turned off, justifying also automatically stops, and justifying resumes automatically if filling is then resumed, unless of course a .NOJ has been received in the interim.

Descriptions of Details of Some Commands

Here are additional comments for some commands:

HEADER, FOOTER -- the header and footer are the next line in the input file in each case. They are read in as is. When it comes time to print them out, they are printed out as is, but with one exception. If the current pseudo-blank character (initially the number sign "#", or ASCII 35) is encountered it is taken as a signal to print at that point a one, two, or three digit page number. Note that as the pseudo-blank character is changed the Header line may have to be reissued. Note that the commands .TM and .BM should be given before the header commands .HE and .FO, if the default sizes of the top and bottom margins are not being used.

.NOFILL -- can also be written .NF

.NOJUSTIFY -- can also be written .NJ

.NUMBER -- sets the number of the next page to be printed, without causing a page break, useful for skipping a page where an illustration will be inserted.

.PARAGRAPH -- note that a negative paragraph indent can be used to get a "hanging indent" -- very useful for lists of items. This page is set up in that way, using a left margin of 10 and a paragraph indent of -10.

.PAGE -- sets the page number to the stated value. If no numeric value is given, the page number will simply increment by 1 each time.

IMPORTANT NOTE

Lines being typed out are modified in two ways, aside from those mentioned. If the pseudo-blank character, initially the "number sign" (ASCII 35) is encountered, it is not printed out but a blank is printed instead. Thus you can force a blank in a way which the program will not recognize as a blank until printout. So if you want a string of characters put out as one word - INCLUDING IMBEDDED BLANKS JUST AS YOU TYPE THEM - then use the pseudo-blank instead of the usual space bar, and the program will consider the whole thing as if one word, and will neither end a line in the middle of it nor insert extra blanks for justifying.

If filling is turned on, whenever a period ".", a question mark "?", or an exclamation point "!" is followed by a blank in the input, an extra blank will be inserted, on the theory that this is the end of a sentence (unless we are at the end of a line). You will notice this in this manual, which was produced using Small Runoff.

EXAMPLE

You will find a sample input file and the result of applying Runoff to it below (separated by dashes). (Note that this sample input has been shifted right by one character so that the Runoff run that produced this document will not affect it -- if you want to extract it and use it as sample input, remove the first blank on each of its lines).

```
--------------------------------------------------------------------
.RM 40
.PL 25
.HE 1
                             Example, p. #
.C 2
A SIMPLE EXAMPLE
= ====== =======
.BL 2
.P 6
This is an example of the kind of file
that we might encounter and want to use Runoff
on. Some of the features we are using are:
.NOJ
.LM 10
.P -5
1. Paragraph indentation.
.P
2. Changing the left margin.
.P
3. Making lists of items like this.
.P
4. Turning right-justification on and off.
.LM 0
.P 5
.J
Note the way we made a list of items by turning
off justification,
.UL 2
moving the left margin in, and
then using a negative indent
(which we have just
reset).
.PG
--------------------------------------------------------------------
```

```
-----------------------------------------------------------------
                                Example, p. 1


          A SIMPLE EXAMPLE
          = ====== =======


     This  is an example of the kind of
file that we might encounter and want to
use Runoff on.  Some of the features  we
are using are:

     1.  Paragraph indentation.

     2.  Changing the left margin.

     3.  Making lists of items like
          this.

     4.  Turning right-justification on
          and off.


                                Example, p. 2


     Note   the   way   we   made   a list of
items   by   turning   off    justification,
moving  the  left  margin  in,  and  then
using  a  negative  indent  (which   we   have
just reset).




-----------------------------------------------------------------
```

Conclusion

The use of line-by-line read and write statements in the program results in a lot of disk activity. Felsenstein has changed the Pascal versions to use a large (say 4k) input buffer which is filled before processing and output begins. This reduces disk head motion and may be faster on some systems. In the future Felsenstein plans to use the UCSD/Pascal MT+ nonstandard procedure BLOCKREAD to speed things up more.

Britten retained the line-by-line read/write activity in the Small version, but speed turned out to be much less of a problem. In one benchmark test a previous version of this paper took 2:00 (minutes:seconds) using the buffered Pascal MT+, and 0:45 using unbuffered Small. The above times are for a 2MHz 8080 with a Shugart Winchester disk. The same test was run with PerSci floppy disk drives, with Pascal MT+ taking 2:22, and Small taking 1:13. The speed and space advantage of Small is due primarily to its deliberate simplicity in design and implementation.

Thanks to Fred Crary and Dick Curtiss for pointing out a number of inconvenient features and bugs.

Let us know of any other bugs you find. We hope that this program can become a group software development project of the Northwest Computer Society, particularly the CP/M-Pascal Users Group. But let's try to keep our individual versions as compatible with each other, and with other versions of Runoff, as possible.

Joe Felsenstein
543-0150 / 365-3829

Randyl Britten
828-8080 / 522-1736

References

1. C. R Britten, "Small - A Simple Mobile Algorithmic Language," NCC'78 Personal Computing Digest, Anahiem 1978, pp 251-271.

2. B. W. Kernighan, P. J. Plauger, Software Tools, Addison-Wesley 1976.