Addresses III

by Jeffreys Copeland and Haemer

Welcome to the fifth in our series about office problems. We're sharing some tools that we've developed over the years to keep us organized, as well as

explaining some of the design decisions that went into building them.

We've spent the past few columns developing tools for handling address books. Because we've arranged our address book as a series of paragraphs delimited by blank lines, the set of tools we've developed will be useful in any database constructed the same way, such as a library card catalog. As usual, we left you with several exercises. Let's look at them now.

Problems and Solutions

Last column, one problem we looked at was how to reverse names–how to turn "Zoe Haemer" into "Haemer, Zoe" to let us sort our address book by surname. After we showed a sed script to do the inversion, we suggested you try converting it to Perl.

The problem is a little more complicated than just splitting a line on a blank and reversing the two halves, because the syntax of names varies a lot. In sed, the solution we gave last time was:

```
#! /bin/sh
```

Jeffrey Copeland (copeland@alumni.caltech.edu) *is a member of the technical staff at QMS's languages group, in Boulder, CO. His recent adventures include internationalizing a large sales and manufacturing system and providing software services to the administrators of the 1993 and 1994 Hugo awards. His research interests include internationalization, typesetting, cats and children.*

Jeffrey S. Haemer (jsh@canary.com) *is an independent consultant based in Boulder, CO. He works, writes and speaks on the interrelated topics of open systems, standards, software portability and porting and internationalization. Dr. Haemer has been a featured speaker at Usenix, UniForum and Expo Kuwait.*

```
sed -e "s/ \([a-z]*\) / \1_/" \
    -e "s/\(.*\) \([-_a-zA-Z\´]*\)\$/\2, \1/" \
    -e "s/_/ /" $*
```

Here's one way to do it in Perl:

```
#! /usr/local/bin/perl -ln
# This does all the original names right.
s/\s([a-z]*\s*\S+\s*$)//;
print "$1, $`";
```

Notice that Perl lets us wrap a connecting article into the surname more easily than we could in sed because of the way it collects its regular expressions.

We also suggested that you enhance the script to handle even odder names, such as Fiorello de la Guardia, Richard P. Feynman, Ph.D. and Martin Luther King, Jr., which either have two articles or a following qualifier. Here's a solution in Perl:

```
#! /usr/local/bin/perl -ln
#
# version to handle extra credit # names

s/\s([a-z\s]*\S+)\s*(,\s*\S+\s*)*$//;
$first = $`;
$last = $1;
$etc = $2;
$first =~ s/\s*$//;
print "$last, $first$etc"
```

This still doesn't deal with "Werner Von Braun." Our filter would have to understand that "Von" is an article, not a middle name. That is, our program would have to stop just looking at simple syntax and start to be aware of the semantics of specific words.

For our own amusement, we went on to write a few other versions. Listing 1 shows one in Tcl, John Ousterhout's tool command language, and our hammer of the month. ("If all you have is a hammer, everything looks like a nail."–Anonymous)

In spite of some linguistic differences, the script's basic structure is remarkably like that of our earlier Perl script–in particular, the regular expression is almost identical.

Long ago, every UNIX utility that handled regular expressions had its own slightly different syntax for those REs. This was fixed by the POSIX.2 standard, which trimmed all the different kinds down to two: basic REs, roughly like those formerly recognized by grep, and extended REs, roughly those formerly used by egrep. The bottom line here is that regular expressions are ubiquitous in UNIX (UNIXbiquitous?), and once you become comfortable with them, they'll be friends for years to come.

Listing 1

```
#!/usr/contrib/bin/tclsh
#
# A comparable tcl routine
#   Despite the differences, note how similar the approach is

set s " \[ \t]"
set S "\[^ \t]"
set pattern      "$s+((\[a-z] |$s)*$S+)$s*(,$s*$S+$s*)*$"
# in Perl "\s([a-z\s]*\S+)\s*(,\s*\S+\s*)*$";

set infile [open [lindex $argv 0]]

while {[gets $infile line] != -1} {
  if {[regexp -- $pattern $line match last ignore etc]} {
    regexp -indices $pattern $line bound_0 bound_1 bound_2 bound_3]


    set all [string trimright $line "\n"]
    set start_1 [lindex [split $bound_1] 0]
    set first [string range $all 0 [expr $start_1-1]]
    set first [string trimright $first $s]
    if {[info exists etc]} {
        set first "$first$etc"
    }
  }
  puts "$last, $first"
}
exit
```

(Just to give you a feel for what things used to be like, Don Libes' Expect, a wonderful new tool built on top of Tcl, has its own, quirky, regular expression semantics that you can memorize if you want to.)

When we wrote a version in awk, we got something different:

```
#!/usr/bin/awk -f
#
# Here's a completely different approach

{
        last = first = etc = " "
        if (split($0, Name, ",") == 2)
            etc = ", " Name[2]

        NF = split(Name[1], Name)
        last = Name[NF]
        for (i = NF-1; i > 0; i--) {
            if (Name[i] ~ /[A-Z]/)
                break
            last = Name[i] " " last
        }
        for (j=1; j <= i; j++)
            first = first " " Name[j]

        print last ", " first etc
}
```

As it turns out, awk doesn't have quite the same array of tools for splitting strings based on regular expressions. Notice, however, that we can still *do* the job. It's just that the approach changes.

Fine and dandy, but how well do each of these different versions perform? Before answering that, we'll throw in one last script that takes away the extra shell script in our original:

```
#!/usr/bin/sed -f
s/ \([a-z]*\) / \1_/
s/\ (.*\) \([-_a-zA-Z\´]*\)$/\2, \1/
s/_/ /
```

With this array of programs, here are some comparison times:

names.awk:	0.2
names.pl:	0.14
names0.sed:	0.48
names0.sh:	0.52
names0.pl:	0.09
names.tcl:	0.36

As you can see, the perl scripts are the fastest. One of these, names0.sh, was our first cut, which doesn't do the extra-credit names correctly. This reminds us that you can make any program faster if you don't care if it works. The other names0 scripts share the same problem and the same speed advantage.

Next comes the awk program, at about half the speed. The Tcl program is slower still. This speed difference is because Tcl is a purely interpreted language, while awk and Perl each have a precompile step. Our original shell script is the slowest, but we can see that most of this is actually the fault of sed itself.

In fact, because Perl is often faster than some of its predecessors, the standard Perl distribution comes with a set of tools for turning other scripts into Perl programs.

Before we leave this topic, we'll ask you to step back and notice that timing data are often very useful. How exactly did we get these particular numbers? Why, with a program, of course. We'll ask you to try your hand at generating some timing data yourself, and show you how we got these next month.

The Missing Program

Last month's script for printing our address book–involving one of our "here are the troff macros; trust us" tricks–required a program called precol, which pre

Listing 2

```
#! /bin/sh
# Bunch up lines into blank-delimited
# paragraphs which fit on a page.

TMPFILE=/tmp/$$.awk"
trap "rm -f $TMPFILE" 0 1 2 3 15

# default page length
len=64

# process arguments
while [ -n "$1" ]
do
   case $* in
   -l*=*)  len=`expr $1 : '-l.*=\([0-9]*\)´`; sh:.ft;;
   -l*[0-9]*)  len=`expr $1 : '-l[^0-9]*\([0-9]*\)´`; shift;;
   -l*)    len=$2; shift; shift;;
   *)      [ -r $1 ] && files="$files $1";
     [ -r $1 ] || echo error: unrecognized flag $1 >&2;
           shift;;
     esac
done
```

vented our paragraphs from being broken across pages by supplying extra blank lines at the end of each page. We promised to show you that program. Here goes.

As usual, we begin by setting some defaults and processing command line arguments. In this case, we only need one flag: page length (see Listing 2).

What's *trap*? That's a shell internal command that takes a certain action if the shell exits with any of the listed signals.

What's `trap`? That's a shell internal command that takes a certain action if the shell exits with any of the listed signals. In this case, if we get signals 0, 1, 2, 3 or 15, we remove a temporary file. What signals are these? We leave this as another exercise to the reader, but suggest you try the command `kill -1`. We allow three different variations for the page length flag: `-l=64`, `-l64` or `-l 64`. We actually recognize anything beginning `-l`, so we could use `-len` or `-length` or `-larry`. This large range of allowable options is probably overkill.

The next thing we must do is prepare the program itself. We do this in awk.

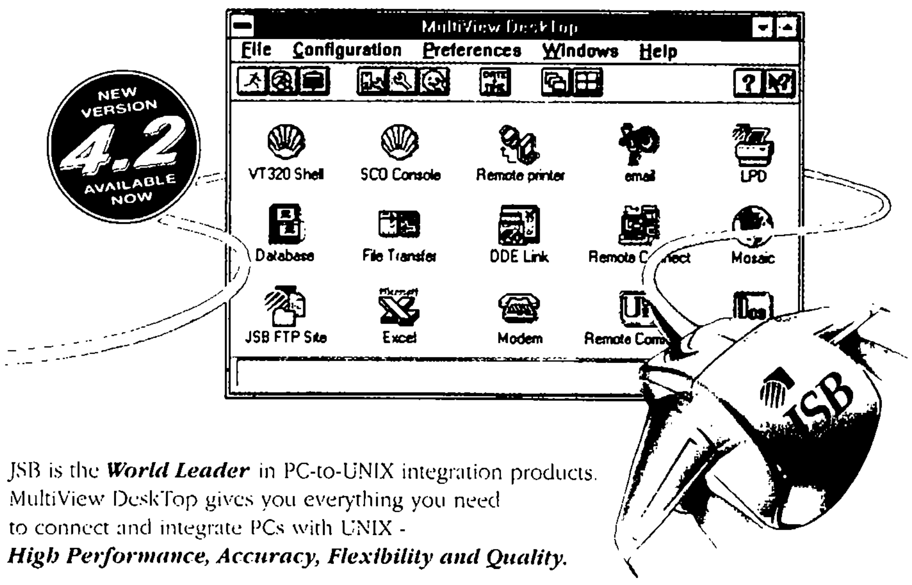
```
# prepare the awk script
cat >$TMPFILE<<EOF
BEGIN  { n = 0; curr = 1; len=$len }
EOF

cat >$TMPFILE<<"EOF"
        .
        .
        .
EOF

# now run the awk script, setting up parameters
# on the command line
awk -f $TMPFILE $files
exit 0
```

Notice that we need to go to some lengths to get the `$len` definition into the script. We could have con-

FLY BETWEEN WINDOWS & UNIX WITH MULTIVIEW DESKTOP



Terminal Emulation
- SCO, Solaris, AT&T, Interactive and UNIXWare Console, WYSE 60, DEC VT100, 220, 241, 320, 420

File Transfer & Print Sharing
- Fastest Drag and Drop FTP
- Kermit
- LPR/LPD Printing
- Transparent Printing

Connectivity
- Includes FREE WinSock TCP/IP and support for 50+ other transports inc. Netware, Serial, Oslan & DECnet

Multisession Support via
- Serial Connections
- IPX/SPX (NVT1 & NVT2)
- TCP/IP

Flexible Configuration
- Easy-to-use and secure desktop
- Floating palette with toolbar, status bar and configurable button bars
- Advanced scripting using tools such as Visual Basic
- Full Keyboard Mapping
- Powerful DDE links

Future-proof legacy networks
- Execute industry-standard Internet applications on existing TCP/IP stacks via 100% WinSock compliant DLL

Options
- UNIX Host Support Server™
- X-Server
- NFS

JSB

JSB is the ***World Leader*** in PC-to-UNIX integration products. MultiView DeskTop gives you everything you need to connect and integrate PCs with UNIX -
High Performance, Accuracy, Flexibility and Quality.

Call 1-800-359-3408 Now for a FREE demo pack

JSB Corporation, 108 Whispering Pines Drive Suite 115, Scotts Valley, California, USA 95066.
Telephone. (408) 438 8300 Fax. (408) 438 8360 email: info@jsb.com www: http://www.jsb.com

All trademarks are acknowledged

Circle No. 7 on Inquiry Card

structed the script so that it was read into /tmp/$$.awk in one lump, but then we would have been forced to escape every dollar sign, except the one in $len. Remember that the environment variables in a shell document are expanded, unless the EOF marker is quoted or escaped.

The "main" routine in our program is the first thing we need to fill into the ellipsis above:

```
/./ { A[n++] = $0 }
/^$/ { showlines() }
END { showlines() }
```

We've buried the interesting part of the program under the showlines() routine: We read lines into an array until we reach a blank line–our paragraph separator–then invoke showlines(). To ensure that we've shown the last paragraph, we invoke showlines() again at the end of the input file.

We're using the version of awk described in *The AWK Programming Language*, by Aho, Weinberger and Kernighan (Addison-Wesley, 1988, ISBN 0-201-07981-X). On a few non-AIX systems, this version is installed as nawk (new awk), and awk invokes an older version that doesn't, for example, even support subroutines.

Of course, we need to define showlines():

```
function showlines() {
    if( n == 0 ) return;
    if( n > len ) {
      printblock();
      curr = (curr + n - 1) % len + 1;
    } else if( curr + n > len ) {
      nextpage();
      printblock();
    } else {
      printblock();
    }
    if( curr > 1 && curr < len ) {
      print
      curr++
    }
}
```

We also still need a pair of utility routines. nextpage() puts out blank lines to space us to the next page, and printblock() prints our buffered paragraph.

```
function nextpage() {
    for( ; curr <= len; curr++ ) print
    curr = 1;
}

function printblock() {
    for( i = 0; i < n; i++ ) print A[i]
    curr += n;
    n = 0;
}
```

Leftovers

We still haven't handled the real problem we left last month–how to select a subset of an address book by geographic location. For example, how do I look for people in the Boston area, so I can have dinner with them on my trip there next week? We'll postpone solving this until next month's column, mostly because we've already gone on too long in this month's, but let's think a bit about the overall problem in the time we have left.

Neither Jeff has a Geographic Information Service database on his machine, so we guess that you may not either.

In the best of all possible worlds, we'd have a directed graph of every city in the world, so that we could say, "Give me a ring of cities three deep with Boston in the center, then grep for the cities on that list." This isn't that world. Still, we already have some useful geographic information in our address book; both postal ZIP code and telephone area code are tied to geographic areas. Unfortunately, metro Boston has 65 ZIP codes.

Alternately, we could find all phone numbers within the area code 617, but if we were visiting Los Angeles, we'd need to look for phone numbers in area codes 213, 818 and 310. Similarly, Manitoba's area code of 204 covers the area from the U.S. border to the Arctic Circle, not just the city of Winnepeg. And neither Jeff has a Geographic Information Service database on his machine, so we're guessing that you may not either.

But what about the Internet? Try this: telnet martini.eecs.umich.edu 3000. Here's the kind of thing you should see:

```
. boulder, co
0 Boulder
1 08013 Boulder
2 CO Colorado
3 US United States
R county seat
```

```
F 45 Populated place
L 40 00 5 N 105 16 12 W
P 76685
E 5344
Z 80301 80302 80303 80304 80306
Z 80307 80308 80309 80310 80314
Z 80322 80323 80328 80329
```

For next month, think about how you might write an application to use the data available from Net sites such as this one.

Warning: *LO* Not Defined

In our first column in this series, we explored building letters in `troff`. Several readers have sent us email asking where they can get the letter macros that we used. The answer is simple: They're part of the standard AT&T -mm macro package. If you're an AIX user you already have them. Unfortunately, some older versions of -mm and the -mgm package for `groff` don't have the letter macros. One of us has hacked together a version that works for `groff`, and the other has converted it to work with `troff` and older versions of -mm. Drop us a note if you need them.

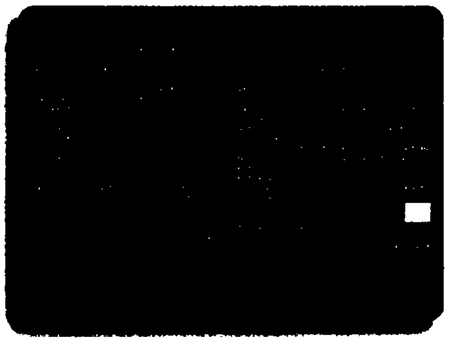
Correction

Also, while we're following up, we had a letter from I. Berelowitz at SIAC in Brooklyn, who discovered an error in our June column. The code for the `crunch` program managed to get garbled in the typesetting, and we didn't catch it in the proofreading. Mr. Berelowitz gets this month's Attention to Detail award for correctly reconstituting `crunch` as

```
#! /bin/sh
# Crunch blocks of multiple lines
# separated by a blank line onto a
# single line, for sorting, grep-ing, etc
awk '/./ { printf "%s\\", $0 }
/^$/ { printf "\\\n" }
END { printf "\\\n" } ' $*
exit 0
```

or even better:

```
#! /bin/nawk -f
/./ { printf "%s\\", $0
/^$/ { printf "\\\n" }
END { printf "\\\n" }
```
▲



Dumb Terminal. Smart Dumb Terminal.

Get The Power Of Multi-tasking With MultiView Mascot.

MultiView Mascot software transforms any ASCII terminal into a smart terminal, with all the benefits of Windowing. Multi-tasking. Window sizing. Cutting and Pasting between applications. Even hot key switching from screen to screen. And you can do it all without modifying your existing software or buying expensive new hardware. MultiView Mascot is also easy to use, complete with a Motif-style interface. So now you can boost productivity immediately and end the frustration of working on dumb terminals.

Winner X/Open Software Supplier of the Year Award 1993

No wonder MultiView Mascot is the world's leading windowing software for dumb terminals. Call **1-800-JSB-UNIX** for more information and a free evaluation copy of MultiView Mascot. ***It's the smart thing to do for dumb terminals.***

JSB Corporation, 108 Whispering Pines Drive, Suite 115, Scotts Vally, California, USA 95066
Telephone (408) 438-8300 Facsimile (408) 438-8360

JSB

All products are trademarks or registered trademarks of their respective holders

Circle No. 8 on Inquiry Card