Fall 2004 Handout 2

A multi-line awk argument

Each line of input has a person's name. The **+2 -3** arguments of **sort** sort by last name; the **+0 -1** break ties by first last name; and the **+1 -2** arguments break further ties by middle name.

Unfortunately, not every line of input has three names. We use **awk** to supply the dummy word **zz** for each missing name. Can you consolidate the **NF == 3** and **NF > 3** lines into a single line?

```
#!/bin/ksh
#Input a list of names, one per line.
#Output them in order of last name, first name, middle name.

awk '
    NF == 3 {printf "%s %s %s", $1, $2, $3}
    NF == 2 {printf "%s zz %s", $1, $2}
    NF == 1 {printf "zz zz %s", $1}
    NF == 0 {printf "zz zz zz"}
    NF > 3  {printf "%s %s %s", $1, $2, $NF}
            {print " @" $0}
' |
sort -f +2 -3 +0 -1 +1 -2 |
awk -F@ '{print $2}'

exit 0
```

Before the first **awk**, the data looks like this:

```
Madonna
Johann Sebastian Bach
George Herbert Walker Bush
Buster Keaton
```

After the first **awk**, the data looks like this:

```
zz zz Madonna @Madonna
Johann Sebastian Bach @Johann Sebastian Bach
George Herbert Bush @George Herbert Walker Bush
Buster zz Keaton @Buster Keaton
```

After the **sort**, the data looks like this:

```
Johann Sebastian Bach @Johann Sebastian Bach
George Herbert Bush @George Herbert Walker Bush
Buster zz Keaton @Buster Keaton
zz zz Madonna @Madonna
```

After the last **awk**, the data looks like this:

```
    Johann Sebastian Bach
    George Herbert Walker Bush
    Buster Keaton
    Madonna
```

Another example of an awk multi-line argument

```
#!/bin/ksh
#Input a list of names, one per line.  Copy each line into one or
#more of the three output files, depending on its first letter.
#You can remove the $0’s.  See KP p. 130, loose end 1.

cd

awk ’
    /^[A-K]/ {print $0 > "names.ak"}
    /^[L-Z]/ {print $0 > "names.lz"}
    /^[J-M]/ {print $0 > "names.jm"}
’

exit 0
```

▼ **Homework 2.1: pinpoint Soviet unrest**

The file **$d46/ussr** contains one line for each ex-Soviet republic:

```
1$ awk ’NR == 1 || $1 == "Belarus" || $1 == "Ukraine"’ $S46/ussr
name        % of land   % of GNP    % of pop.   population
Belarus     1.0         4.2         3.5          10200000
Ukraine     2.7         16.2        18.0         51704000
```

As you see, the White Russians have more income per capita than their hereditary enemies, the Ukranians. Write a shellscript named **unrest** that will output into a file named **uppers** in your home directory the name of every republic whose citizens have a greater than average per capita income. (Four of the fifteen republics listed are in this enviable position.) Simply **print** the names of the republics where field three is greater than field four. Also create an output file named **downers** for the ten republics whose citizens have a less than average per capita income. Create an **average** output file, too.

Print the three output files side by side:

```
2$ pr -l1 -m -t uppers downers average                  minus lowercase L one
Belarus                 Armenia                 Estonia
Latvia                  Azerbaijan
Lithuania               Georgia
```

▲

Another example of an awk multi-line argument

Add an optional command line argument to the following shellscript to let the use specify the interval between dividing lines (default 10).

```
#!/bin/ksh
#This shellscript, named divide, outputs a copy of its input with a
#dividing line inserted every 3 lines.
#Sample use: 1$ divide < /etc/passwd

awk '
                 {print}
    NR % 3 == 0  {print "------------------------------"}
'
exit 0
```

```
2$ divide < /etc/passwd | head -13
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
------------------------------
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
------------------------------
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
sshd:x:17:17:Sshd:/var/empty:/bin/false
------------------------------
listen:x:37:4:Network Admin:/usr/net/nls:
```

A multi-statement action

Suppose each assembly language opcode comes in five sizes. To list them all without typing each one five times, we let **awk** add the five extensions for us. For example, if we input this file

```
add
mov
sub
```

to any of these three shellscripts,

```
#!/bin/ksh
#Output each opcode with all five extensions.

awk '{print $1 "b\n" $1 "w\n" $1 "l\n" $1 "f\n" $1 "d"}'
exit 0
```

```
#!/bin/ksh
#Output each opcode with all five extensions.

awk '
    {
        print $1 "b"     #byte
        print $1 "w"     #word
        print $1 "l"     #long
        print $1 "f"     #float
        print $1 "d"     #double
    }
'

exit 0
```

```
#!/bin/ksh
#Output each opcode with all five extensions.

awk '{print $1 "b"; print $1 "w"; print $1 "l"; print $1 "f"; print $1 "d"}'
exit 0
```

the output is

```
addb
addw
addl
addf
addd
movb
movw
movl
movf
movd
subb
```
etc.

▼ Homework 2.2: output the initialization for a C array of strings

Modify the above shellscript to output each opcode surrounded by double quotes and followed by a comma. Output a numbered C comment (starting at 0) alongside the first opcode in each group. Output an empty line after the last opcode in each group. The input file should be the one shown above, and the new output will be

```
"addb",     /* 0 */
"addw",
"addl",
"addf",
"addd",

"movb",     /* 1 */
"movw",
```
etc.

Write another shellscript to output the initialization for a C array of structures, each containing five strings:

```
{"addb", "addw", "addl", "addf", "addd"},       /* 0 */
{"movb", "movw", "movl", "movf", "movd"},       /* 1 */
{"subb", "subw", "subl", "subf", "subd"},       /* 2 */
```
etc.

▲

An if statement in an action: pp. 119−122

```
#!/bin/ksh
#Output the last name of everyone in the class, one per line.

for file in ˜mm64/public_html/x52.9545/bio/*
do
    awk 'NR == 2 {print $NF}' $file
done

exit 0
```

```
#!/bin/ksh
#Output the last name of everyone in the class, one per line.
#If the name ends with "Jr.", print the next-to-last word instead.

for file in ˜mm64/public_html/x52.9545/bio/*
do
    awk '
        NR == 2 && $NF != "Jr." {print $NF}
        NR == 2 && $NF == "Jr." {print $(NF - 1)}
    ' $file
done

exit 0
```

To avoid duplication, write it as one action containing an **if** statement. You can use the C **if**, **else**, **for**, **while**, **do-while**, **continue**, and **break** statements within the curly braces of an **awk** action. Within the curly braces of an **awk** action, the rules for curly braces are the same as those in the language C.

```
#!/bin/ksh
#Output the last name of everyone in the class.
#If the name ends with "Jr.", print the next-to-last word instead.

for file in ˜mm64/public_html/x52.9545/bio/*
do
    awk '
        NR == 2 {
            if ($NF == "Jr.") {
                print $(NF - 1)
            } else {
                print $NF
            }
        }
    ' $file
done

exit 0
```

An assignment statement in an action

```
#!/bin/ksh
#Output the colors in /usr/openwin/lib/rgb.txt in order of how
#close they are to 200 100 50

awk '
    NR >= 2 {
        red = $1 - 200
        green = $2 - 100
        blue = $3 - 50
        print red*red + green*green + blue*blue, $0
    }
' /usr/openwin/lib/rgb.txt |
sort -n |
sed 's/^[1-9][0-9]* //'

exit 0
```

```
    205 104  57     sienna3
    205  91  69     coral3
    205 102  29     chocolate3
    205  79  57     tomato3
    210 105  30     chocolate
```

▼ Homework 2.3: parameterize the color

Instead of hardwiring the color (e.g., **`200`**, **`100`**, **`50`**) into the above shellscript, let the user specify it as three command line arguments.
▲

Search using user-defined variables

```
#!/bin/ksh
#Read in lines and output one copy of the line that appeared most
#frequently.  If two or more lines are tied, output one copy of each
#of those that are tied.

sort |
uniq -c |
sort -nr |
awk '
    NR == 1 {n = $1}
    $1 == n
' |
sed 's/^ *[1-9][0-9]* //'
exit 0
```

To output the line(s) that appeared least frequently, change **`-nr`** to **`-n`**.

▼ Homework 2.4: list the title of each homework for X52.9545

In the **`$m45/handout/handout*.ms`** files, the title of each homework is preceded by the **`.HW`** macro defined in **`$m45/handout/header.ms`**. For example, I typed

```
.HW
list the title of each homework for X52.9545
.PP
In the
```

Write a shellscript named **hwindex** that will output every line in the **$m45/handout/handout*.ms** files that immediately follows a **.HW** line. Give the twelve input filenames to **awk** as command line arguments in numerical order:

```
cd $m45/handout
awk 'first argument goes here' `ls handout*.ms | sort +0.7n`
```

On January 9, 2004, the correct output was

```
1$ hwindex | head
interesting but harmless things to do
chmod a directory
create three directories
draw part of the tree
create, print, and copy a file
get three people's email addresses
how many files are there?
mail a letter to yourself and read it
search for duplicates
find duplicate programs
```

▲

Arithmetic in awk: KP pp. 118−119

```
#!/bin/ksh
#Input a list of numbers, one per line, and output their sum.
#The BEGIN line is unnecessary here and is always omitted: p. 118.

awk '
    BEGIN   {sum = 0}
            {sum = sum + $1}
    END     {print sum}
'
exit 0
```

```
1$ lpq
nyu_acf_th_hp8150_1 is ready and printing
Rank   Owner      Job  Files                                 Total Size
active abc1234    42   moe                                    29 bytes
1st    def5678    43   larry                                 100 bytes
2nd    def5678    44   (standard input)                      512 bytes
```

```
#!/bin/ksh
#Output the total number of bytes in all the files waiting to be
#printed on the specified printer.
#Sample use: 2$ waiting -Pth_hp8150_1

if [[ $# -ne 1 ]]
then
    echo $0: requires exactly one command line argument 1>&2
    exit 1
fi

if [[ $1 != -P* ]]
then
    echo $0: command line argument must start with -P 1>&2
    exit 1
fi

lpq $1 |
awk '
    /bytes$/ {sum = sum + $(NF-1)}
    END      {print sum}
'
exit 0
```

```
#!/bin/ksh
#Print the files specified as command line arguments on the printer
#whose queued files total the smallest number of bytes.
#Sample use: 3$ fastest file1 file2 file3

if [[ `waiting -Pth_hp8150_1` -lt `waiting -Pnd_hp3si_3` ]]
then
    echo printing $* on th_hp8150_1
    lpr -Pth_hp8150_1 $*
else
    echo printing $* on nd_hp3si_3
    lpr -Pnd_hp3si_3 $*
fi

exit 0
```

Total and average size in bytes of all the files in the current directory

```
1$ cd $m46/handout
2$ ls -la | head -5
total 3176
drwxr-xr-x   5 mm64     users       8192 Nov 21 13:45 .
drwxr-xr-x   6 mm64     users         96 Jun 19  2002 ..
-r--r--r--   1 mm64     users       4930 Nov 23  1994 adb.ms
drwxr-xr-x   2 mm64     users       4096 Jan 31  2001 answer
```

```
#!/bin/ksh
#Output the average size in bytes of all the files in the current
#directory.

ls -la |
tail +2 |
awk '
            {sum = sum + $5}
    END     {print "total:", sum, "average:", sum/NR}     #bug
'
exit 0
```

```
#!/bin/ksh

ls -la |
tail +2 |
awk '
    /^-/    {sum = sum + $5}
    END     {print "total:", sum, "average:", sum/NR}     #bug
'
exit 0
```

```
#!/bin/ksh

ls -la |
tail +2 |
awk '
    /^-/    {sum = sum + $5; count = count + 1}
    END     {print "total:", sum, "average:", sum/count}  #bug
'
exit 0
```

```
#!/bin/ksh

ls -la |
tail +2 |
awk '
    /^-/    {sum = sum + $5; count = count + 1}
    END     {
                if (count > 0) {
                    print "total:", sum, "average:", sum/count
                } else {
                    print "total:", sum
                }
            }
'
exit 0
```

Abbreviate **count = count + 1** and **sum = sum + $5** as in C. See pp. 118 and 121 for **+=** and **++**.

▼ Homework 2.5: cpu time

`+0.46 -0.49` tells **`sort`** to sort in order of minutes; **`+0.50 -0.52`** tells **`sort`** to break ties in order of seconds.

```
1$ ps -Af | sort -nr +0.46 -0.49 +0.50 -0.52 | more
     UID   PID  PPID  C    STIME TTY      TIME CMD
  nobody   472   463  0   Nov 07 ?      111:58 /usr/local/apache/bin/httpd
  nobody   676   463  0   Nov 07 ?      109:06 /usr/local/apache/bin/httpd
  nobody   473   463  1   Nov 07 ?      101:35 /usr/local/apache/bin/httpd
  sunnet   385     1  0   Nov 07 ?      100:14 /opt/SUNWsrspx/bin/srsproxy
  nobody   671   463  0   Nov 07 ?      110:24 /usr/local/apache/bin/httpd
  nobody   655   463  0   Nov 07 ?      107:14 /usr/local/apache/bin/httpd
```

```
#!/bin/ksh
#Output the CPU time in seconds of each process.

ps -Af |
awk 'NR >= 2 {print 60 * substr($0, 47, 3) + substr($0, 51, 2)}'

exit 0
```

```
0
0
0
0
0
0
```

Write a shellscript named **`cpukill`** that will **`kill -9`** every program that has used more than 100 seconds of cpu time, unless the program's name is **`-sh`**, **`-csh`**, **`-ksh`**, or **`vi`**. Pipe the output of **`ps -Af`** into an **`awk`** command that outputs the **`PID`** number of each program to be killed. Then use back quotes.

✎ Extra credit. Also have **`cpukill`** mail a one-line letter to the owner of each slain program. Have **`awk`** output to a temporary file the login name of each owner. Then after **`awk`** is finished running, mail a letter to each of these people and remove the temporary file.
▲.

Arrays in awk: pp. 122−123

See also the **`backwards`** example in KP p. 122.

```
#!/bin/ksh
#Input a list of numbers, one per line, and output their median.
#Begin by sorting the numbers.  If there is an odd number of numbers,
#then the median is the middle number.
#If there is an even number of numbers, then the median is the
#average of the two middle numbers.

sort -n |
awk '
          {a[NR] = $0}    #Copy each line into an array.

     END {
          if (NR == 0) {
               print 0                                  #none
          } else if (NR % 2 == 0) {
               print (a[NR/2] + a[1 + NR/2])/2          #even
          } else {
               print a[(NR + 1)/2]                      #odd
          }
     }
'

exit 0
```

Two kinds of for loops in awk: pp. 123−124

```
1$ ps -A -ouid,rss,comm | more
  UID  RSS COMMAND
    0    0 sched
    0  184 /etc/init
 1005 7320 /opt/SUNWsrspx/bin/srsproxy
    1 1520 /usr/lib/nfs/statd
50766 1000 awk
 2689  704 sleep
```

```
#!/bin/sh
#Output a table whose first column lists the UID number of everyone
#running a process, and whose second column shows how many K of memory
#all of their processes are using.

ps -A -ouid,rss |
awk '
    NR >= 2 {sum[$1] += $2}

    END {
        for (uid = 0; uid < 65536; ++uid) {
            if (sum[uid] != 0) {
                printf "%5d\t%10d\n", uid, sum[uid]
            }
        }
    }
' |
sort +1nr +0n

exit 0
```

```
    0          221792
60001           22488
50766           13768
 1005            7320
 2689            6176
```

`0` *is the* **`UID`** *number of* **`root`**.

To fix the bug in the above program, we should change the **`NR >= 2`** action

```
{sum[$1] += $2}
```

to

```
{
    if ($2 ~ /K$/) {
        sum[$1] += $2
    } else {
        sum[$1] += 1024 * $2
    }
}
```

The **`for`** loop shown above makes the variable **`uid`** run through all 65,536 possible **`UID`** values.

```
2$ awk -F: '{print $3}' /etc/passwd | sort -nr | head -1
793876

3$ grep uid_t /usr/include/sys/types.h
typedef ushort_t o_uid_t;       /* old UID type     */
typedef o_uid_t o_gid_t;    /* old GID type     */
typedef int uid_t;          /* UID type     */
typedef longuid_t;          /* (historical version) */
typedef uid_t   gid_t;          /* GID type     */

4$ grep uint_t /usr/include/sys/types.h
typedef unsigned int uint_t;
```

```
5$ grep UINT_MAX /usr/include/iso/limits_iso.h
#define UINT_MAX 4294967295U /* max value of an "unsigned int" */
```

To run the variable **uid** through only those subscripts of the array **sum** that have been assigned values, use the other kind of **awk for** loop:

```
#!/bin/sh
#Faster way to do the same thing.

ps -A -ouid,rss |
awk '
    NR >= 2 {sum[$1] += $2}

    END {
        for (uid in sum) {
            printf "%5d\t%10d\n", uid, sum[uid]
        }
    }
' |
sort +1nr +0n

exit 0
```

This kind of **for** loop gives values to **uid** in an unpredicatable order. The **sort**, however, makes this irrelevant.

An associative array in awk

To print the loginnames instead of the **UID** numbers in column 1,

```
1$ ps -A -ouser,rss,comm | more
    USER  RSS COMMAND
    root    0 sched
    root  184 /etc/init
  sunnet 7320 /opt/SUNWsrspx/bin/srsproxy
  daemon 1520 /usr/lib/nfs/statd
   recon  704 sleep
   jf233 1320 /usr/local/bin/perl
```

```
#!/bin/sh
#Output a table whose first column lists the loginname of everyone
#running a process, and whose second column shows how many K of memory
#all of their processes are using.

ps -A -ouser,rss |
awk '
    NR >= 2 {sum[$1] += $2}

    END {
            for (loginname in sum) {
                printf "%-8s\t%10d\n", loginname, sum[loginname]
            }
        }
' |
sort +1nr +0

exit 0
```

```
root            221896
nobody           22552
mm64             13296
sunnet            7320
recon             6176
```

Remove duplicate input lines without disturbing their order

```
sort | uniq
sort -u
```

a faster way to do the same thing

```
#!/bin/sh
#Output a copy of the lines read as input.  If a line appears more
#than once in the input, output only the first copy.  In other words,
#this shellscript does what "uniq" does, but it does not require that
#duplicate input lines be consecutive.

awk '
                    {line[$0] = line[$0] + 1}
    line[$0] == 1   {print $0}
'
```

```
awk '++line[$0] == 1'
```

a faster way to do the same thing

▼ **Homework 2.6: list the size of each user's largest process**

Write a shellscript that starts with **ps -A -ouser,rss,comm**, and outputs the **rss** of the biggest program that each user is running. Create an associative array named **rss** whose subscripts will be the loginnames read from input. Sort the output of **awk** in order of decreasing **rss**, breaking ties (if any) in alphabetical order of loginname.

▲

Collating sequences for sort: pp. 19, 106

The collating sequences include

alphabetical (the default);

`-f` alphabetical, ignoring the difference between upper and lowercase;

`-b` alphabetical, ignoring leading blanks and tabs (works on acf5 but not acf4);

`-d` alphabetical, ignoring dashes, apostrophes, etc.;

```
Oppenheim
O'Reilly
Ostrov
```

ignore case, too: **`-df`**

`-n` increasing numeric, including negative numbers and decimal points, ignoring leading blanks and tabs;

`-M` chronological, but it knows only 12 words: **`Jan`**, **`Feb`**, **`Mar`**, etc. It ignores case and the rest of the word. Sun has this option, but DEC doesn't.

Add an **`r`** to any of the above to reverse the order, e.g., **`-nr`** gives decreasing numeric.

Ignore initial field(s) and break ties

`sort +`*n* ignores the first *n* fields on each line. For example,

```
1$ ls -l | tail +2
-rw-r-----   1 abc1234  users      14866 Oct  5 18:12 file1.c
-rw-------   1 def5678  users      10812 Oct  5 16:36 file10.c
-rw-r--r--   2 abc1234  users      14866 Oct  5 18:13 file2.c

2$ ls -l | tail +2 | sort +2                alphabetically by owner's name

3$ ls -l | tail +2 | sort +4n               increasing size order
4$ ls -l | tail +2 | sort +4nr              decreasing size order

5$ ls -l | tail +2 | sort +4n +2            increasing size order;
                                            break ties (if any) by alphabetical order of owner's login name

6$ ls -l | tail +2 | sort +4n +2 +8         increasing size order;
                                            break ties (if any) by alphabetical order of owner's login name
                                            break further ties (if any) by alphabetical order of filename
```

If two lines are still tied after all the comparisons you have asked for, **`sort`** sorts the two lines in plain vanilla alphabetical order. This happens, for example, when you **`sort -n`** lines that do not start with a number.

Here are two ways to do the same thing:

```
sort +2n +3n +1n
sort -n +2 +3 +1
```

A stand-alone letter such as the above **`-n`** applies to all of the following fields. If this is not what you want, write a separate letter for each field.

▼ Homework 2.7: sort with multiple fields

The three files **`$d46/date1`**, **`$d46/name3`**, and **`$d46/netaddress`** contain lines of the form

```
9/9/95                                  month/day/year
John Philip Sousa
mm64@acf5.nyu.edu
```

respectively. Write three **`sort`** commands to sort them. Sort **`date1`** chronologically, with the oldest dates on top. The **`-t`** option of **`sort`** is just like the **`-F`** option of **`awk`**. Sort **`name3`** by last name; then break ties by first name; then break ties by middle name. Ignore case and dashes and apostrophes in **`name3`**. Sort **`netaddress`** by hostname (e.g., **`i5.nyu.edu`**); then break ties by loginname (e.g., **`mm64`**). Ignore case in the hostname, but not in the loginname.

▲

sed example: sort playing cards in order of increasing rank

Suppose you have a file of playing cards, one per line. The first column is the rank and the second column is the suit:

```
A   S
2   C
Q   H
J   D
```

```
#!/bin/ksh
#Sort playing cards, one per line, in order of increasing rank.
#Ignore the suits.
#Add "14 " to start of every line that begins with "A" or "a".
#For & in an s/// command, see textbook pp. 323-324; Handout 8, p. 7.

sed '
    s/^[2-9]/& &/
    s/^10/& &/
    s/^[Jj]/11 &/
    s/^[Qq]/12 &/
    s/^[Kk]/13 &/
    s/^[Aa]/14 &/
' |
sort -n |
sed 's/^[^ ]* //'   #Remove everything up to and including 1st blank.

exit 0
```

```
#!/bin/ksh
#Sort playing cards, one per line, in order of increasing rank.
#Ignore the suits.
#Add "14 " to start of every line that begins with "A" or "a".

awk '
    /^([2-9]|10)/ {print $1 $0}
    /^[Jj]/       {print 11 $0}
    /^[Qq]/       {print 12 $0}
    /^[Kk]/       {print 13 $0}
    /^[Aa]/       {print 14 $0}
' |
sort -n |
sed 's/^[^ ]* //'      #Remove everything up to and including 1st blank.

exit 0
```

After the first **sed**, our data is:

```
14 A   S
2 2    C
12 Q   H
11 J   D
```

After the **sort -n**, our data is:

```
2 2    C
11 J   D
12 Q   H
14 A   S
```

After the second **sed**, our data is:

```
2   C
J   D
Q   H
A   S
```

▼ **Homework 2.8: sort whatever you want**

Write a shellscript named **customsort** that sorts its lines of input into an order other than alphabetical or numerical. For example, if your input consists of one chemical element per line, sort them in order of increasing atomic number. Just do the first ten elements: **H**, **He**, **Li**, **Be**, **B**, **C**, **N**, **O**, **F**, and **Ne**.

before	*after*
H	H
O	He
He	Li
Li	B
B	O
O	O

▲

▼ **Homework 2.9: chronological sort**

The file **$S46/date2** contain lines of the form

```
February 24, 1996
February 25, 1996
October 29, 1996
```

Write a shellscript to sort them in chronological order, with the oldest date first. If your **`sort`** has no **`-M`** option, prepend a number from 1 to 12 and a blank to each line.

```
2 February 24, 1996
2 February 25, 1996
10 October 29, 1996
```

Then **`sort`** and remove the leading number and blank.
▲

▼ Homework 2.10: a compilation lister

Use the GNU **`gcc`** compiler for this assignment.

```
#include <stdio.h>
main()
{
    int i = 10;
    printf ("%d\n", j);
}
```

```
1$ gcc junk.c >& junk.err
2$ cat junk.err
junk.c: In function main:
junk.c:5: ‘j’ undeclared (first use this function)
junk.c:5: (Each undeclared identifier is reported only once
junk.c:5: for each function it appears in.)
```

See p. 93 for the Bourne shell equivalent of **`>&`**.

Write a shellscript named **`compile`** that will take one **`.c`** file as a command line argument and compile it. You get no credit unless the name of the **`.c`** file is supplied to the shellscript as a command line argument. Sample use:

```
3$ compile junk.c | more
 1:#include <stdio.h>
 2:main()
 3:{
 4:   int i = 10;
 5:   printf ("%d\n", j);
*5: ‘j’ undeclared (first use this function)
*5: (Each undeclared identifier is reported only once
*5: for each function it appears in.)
 6:}
```

Add a zero, a colon, a line number, and another colon to the start of each line of the C program:

```
0:1:#include <stdio.h>
0:2:main()
0:3:{
0:4:    int i = 10;
0:5:    printf ("%d\n", j);
0:6:}
```

Remove error lines that do not have a line number (e.g., **`In function main:`**). Then remove the filename from the start of each surviving error line i.e., remove everything up to but excluding the first colon.

Finally, add an increasing number to the start of each error line:

```
    1:5: 'j' undeclared (first use this function)
    2:5: (Each undeclared identifier is reported only once
    3:5: for each function it appears in.)
```

You get no credit for any shellscript unless you put the temporary files you create into your home directory:

```
    prog1 | prog2 | prog3 > $HOME/program
    prog4 | prog5 | prog6 > $HOME/errors
```

You get no credit if you input a file into a program like this:

```
    cat file | prog
```

Do it this way instead:

```
    prog file
```

Merge the two temporary files by feeding them into **sort -t: -n +1 +0**. Finally, change each leading zero and colon into a blank, and each leading non-zero number (which may have more than one digit) and colon into an asterisk.

Use the smallest number of **awk**'s or **sed**'s to do the job.

▲

Ignore initial character(s)

sort +8.4bn ignores the first eight fields, then ignores the first four characters of the ninth field, and then does a numeric sort on what remains. (Without the **b**, our version of **sort** would count the blank between the eighth and ninth fields as part of the ninth field.)

```
1$ ls -l | tail +2 | sort +8.4bn
2$ ls -l | tail +2 | sort +8.5n
3$ ls -l | tail +2 | sort -k +9.5bn
-rw-r-----   1 abc1234  users      14866 Oct  5 18:12 file1.c
-rw-r--r--   2 abc1234  users      14866 Oct  5 18:13 file2.c
-rw-------   1 def5678  users      10812 Oct  5 16:36 file10.c
```

-k *numbers are one-base*

Restrict the end of a sort field

A numeric sort ignores leading blanks and tabs at the place you direct it to. It pays attention only to the number, and then ignores whatever comes after the number. An alphabetical sort, however, pays attention to every character from the place you direct it to all the way to the end of the line. We say that an alphabetical sort field extends to the end of the line by default.

You can specify where a sort field ends as well as where it begins by using a pair of arguments with a plus and minus. For example, to sort alphabetically by the owner's name and break ties by sorting alphabetically by the filename,

```
    1$  ls -l | tail +2 | sort +2 -3 +8
```

The **+2 -3** work together as a unit. The **+2** means "start paying attention after ignoring the first two fields". The **-3** means "stop paying attention at the end of the third field".

Restrict the end of a sort field to a specific character

To sort social security numbers (nine digits), one per line, in increasing numerical order but ignoring the first three digits,

```
    sort +0.3n
```

To sort social security numbers, one per line, in increasing numerical order but ignoring the first three and last four digits,

```
sort +0.3n -0.5
```

In the following example, the **`+0.0 -0.1`** makes **`sort`** pay attention to the first character only, and the **`+8`** makes **`sort`** break ties by the ninth field. Thus it lists all the files first (since **`-`** comes before **`d`**) in alphabetical order, followed by all the directories in alphabetical order.

```
1$ ls -l | tail +2 | sort +0.0  -0.1 +8          files before directories
2$ ls -l | tail +2 | sort +0.0r -0.1 +8          directories before files
```

▼ Homework 2.11: four-field sort

Sort the loginnames of the form **`abc1234`** in **`/etc/passwd`** in alphabetic order by the last initial (i.e., the third letter); then first initial (the first letter); then middle initial (the second letter); then in increasing numeric order by the four digits from the social security number. You get credit only if you explicitly sort the four digits in numerical order, not in alphabetical order. Give no more than five arguments to **`sort`** by combining the first and middle initials into a single field. Pipe the output of **`sort`** through **`head -50`** into

```
pr -5 -l10 -t                                   minus lowercase L ten
```

to print only the first 50 loginnames, in five columns of ten lines each. You get credit only if you search for loginnames of *exactly* seven characters, no more and no less: three lowercase letters followed by four digits. The first 50 loginnames of this form (as of January 9, 2004) were

```
dqa0772     sqa7460     ejb3500     mlb4008     tqb8179
dqa9188     anb2023     fqb7538     mqb4793     vlb2007
fda6677     aqb4851     gbb0405     mqb5363     acc4707
ima9488     bqb9408     gqb2634     nkb1384     adc2416
jqa7024     ceb6547     izb6225     pjb7346     aqc1982
lma2018     dab3766     jqb2622     rab2017     aqc3484
mka7827     dmb9568     kjb8147     sdb1150     aqc7242
mqa8127     dqb0497     krb6949     skb3062     asc3018
pqa2778     dqb2129     leb0734     smb8818     cdc6040
sqa2554     dqb3240     lzb8099     sqb1811     czc4626
```

▲

□