Appendix C2

Program source of the wordlist compiler, mk_list.awk

```
# ================================================================================
# mk_list.awk (c) Y.Someya with TOHO, August 1, 1998
# ================================================================================
# [Usage-1] jgawk -f mk_list.awk INFILE1 INFILE2 ...INFILEn > NEWFILE
# [Function] Create a consolidated word frequency comparison table from multiple input
# files. The input file is assumed to be a plain running text file and no pre-editing
# is needed.
# [Usage-2] jgawk -f mk_list.awk base=BASEFILE INFILE1 INFILE2 ...INFILEn > NEWFILE
# [Function] Create a consolidated word frequency comparison table for those words
# listed in an existing wordlist (= the base file). To print a separate list of words
# NOT included in the base file, add the "misc" option to the command line, as in the
# following:
# jgawk -f mk_list.awk base=BASEFILE misc=OUTFILE INFILE1 INFILE2 ...INFILEn > NEWFILE
# The base file shall be an one-entry-per-line simple wordlist sorted in alphabetical
# order.
# [Usage-3] jgawk -f mk_list.awk prop=OUTFILE INFILE1 INFILE2 ...INFILEn > NEWFILE
# [Function] Print a normalized frequency (N. Freq.) comparison table. The default N.
# Freq. setting is 10,000 (See Program Block 21). The above command line will print
# an N. Freq. comparison table of the input files 1-n to the OUTFILE, and an absolute
#  frequency table of the same input files to the NEWFILE respectively.

# [Note-1] Instead of typing input file names one by one, you may first store them in
# a temporary directry (e.g. C:\TEMP) and specify all the files in the directry using
# the wild card, as follows:
# jgawk -f mk_list.awk OPTIONS C:\TEMP\*.* > NEWFILE
# [Note-2] The following OPTIONS may be used, but the "prop" and "base" options cannot
# be specified at the same time.
# prop=OUTFILE (Specify a proportional data output file)
# base=BASEFILE (Specify a base file name)
# misc=OUTFILE  (Specify an output file name for words not in the base file)
# case=on (Case sensitive)
# msg=off (Don't print messages)
# head=off (Don't print header)
# total=off (Don't print total)
# ================================================================================
# (1) BUBBLE SORT
function bsort(array, ref, left, right){
   for (i = right; i > left; --i){
      for (j = left; j < i; j++){
         if (ref[array[j]] > ref[array[j+1]]){
            temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
         }
      }
   }
}
# (2) QUICK SORT
function qsort(array, ref, left, right,i, j, key, temp, len){
   len = right - left;
   for (i = int(((totalLen % 1000) + len) / 1000); i > 0; --i){
      if (msg != "off") printf(".") > "/dev/stderr";
   }
   totalLen += len;
   if (len < 1) return;
   if (len == 1){
        if (ref[array[left]] <= ref[array[right]]) return;
        temp = array[left]; array[left] = array[right]; array[right] = temp;
```

```
        return;
    }
    if (len <= 5) bsort(array, ref, left, right);
    key = ref[array[int((left + right) / 2)]];
    i = left;
    j = right;
    while (i < j){
       while (ref[array[i]] < key){ i++; }
       while (key < ref[array[j]]){ --j; }
       if (i >= j) break;
       temp = array[i];
       array[i] = array[j];
       array[j] = temp;
       i++; --j;
    }
    if (i == j){ if (ref[array[i]] < key){ i++; }else{--j; }}
    qsort(array, ref, left, j);
    qsort(array, ref, i, right);
}
# (3) printHeader -- function to print header; "-" is used for the standard input
function printHeader(output, word, label, files, sw, f){
     printf("%s ", word) > output;
     for (f = 1; f <= files; f++){
         printf("%s ", label[f]) > output;
     }
     if (sw) printf("TOTAL\n") > output; else printf("\n") > output;
}
# (4) printLine -- function to print one line
function printLine(output, word, id, freq, files, sw, f, sum){
    printf("%s ", word) >output;
    sum = 0;
    for (f = 1; f <= files; f++){
       if ((id != 0) && ((f, id) in freq)){
          sum += freq[f, id];
          printf("%d ", freq[f, id]) > output;
       }else{
          printf("0 ") >output;
       }
    }
    if (sw) printf("%d\n",sum) > output; else printf("\n") > output;
}
# (5) printData -- function to print one line data
function printData(output, word, data, files, sw, f){
     printf("%s ", word) >output;
     sum = 0;
     for (f = 1; f <= files; f++){
         printf("%f ", data[f]) > output;
     }
     if (sw) printf("%f\n",data[0]) > output; else printf("\n") > output;
}
# (6) printFooter -- function to print vertical total
function printFooter(output, word, number, files, sw, f, sum){
     printf("%s ", word) >output;
     sum = 0;
     for (f = 1; f <= files; f++){
         sum += number[f];
         printf("%d ", number[f]) >output;
     }
     if (sw) printf("%d\n", sum) > output; else printf("\n") >output;
}
# (7) INITIALIZATION
BEGIN {
```

```
  FS = "[ \t\"\\[\\]\\\\,;:(){}<>!?/&+=@#$%^*~`|]";
  fileno = 0;
  wordno = 0;
  sum = 0;
  line = 1;
}
# (8) At the beginning of each file, display messages, count files and
# save the file name.
FNR == 1 {
   # (9) END OF FILE -- DISPLAY NUMBER OF WORDS PROCESSED
   if (fileno != 0){
      if (msg != "off") printf("[ %d lines, %d words, done.] \n", NR - line,
      sum)  > "/dev/stderr";
      vTotal[fileno] = sum;
      line = NR;
      sum = 0;
   }
   # (10) START OF FILE -- SAVE THE FILE NAME AND DISPLAY A MESSAGE
   label[++fileno] = FILENAME;
   if (msg != "off") printf("processing \"%s\" ", FILENAME) > "/dev/stderr";
}
FNR % 100 == 0 { if (msg != "off") printf(".") > "/dev/stderr"; }
# (11) IGNORE COMMENT LINES WITH INITAIL # MARK IN DATA FILES
/^#/ { next
       }
# (12) For each word, increase frequency
{
   for (i = 1; i <= NF; i++){
      word = $i;
# (13) REMOVE PERIODS (".") -- WORD MUST HAVE AN ALPHABET CHAR.
      gsub("[.]","",word);
      if ((word ~ /[A-Za-z]/)){
         if (case != "on") word = tolower(word);
# (14) REGISTER NEW WORDS
         if (!(word in wordlist)){
            wordlist[word] = ++wordno;        # associated list
         }
# (15) INCREMENT COUNT
         id = wordlist[word];
         count[fileno, id]++;
         sum++;
      }
   }
}
# (16) PRINT RESULTS
END {
   vTotal[fileno] = sum;
   if (msg != "off") printf("[ %d lines, %d words, done.] \n", NR - line + 1,
   sum) > "/dev/stderr";
# (17) CHECK OPTIONS
   hTotalSW = 1;
   if ((total  == "off") || ((fileno == 1) && (total != "on"))){
      hTotalSW = 0;
   }
# (18) IN CASE OF NO BASEFILE, PRINT ALL WORDS IN ALPHABETICAL ORDER
   if (base == ""){
# (19) CREATE A SORTED WORDLIST IN WORD[]
      for (word in wordlist) {
         revlist[wordlist[word]] = word;
         delete wordlist[word];
      }
      for (i = 1; i <= wordno; i++){ order[i] = i; }
```

```
      if (msg != "off") printf("sorting %d words ", wordno) > "/dev/stderr";
      qsort(order, revlist, 1, wordno);
      if (msg != "off") printf(" done.\n") > "/dev/stderr";
# (20) PRINT THE TABLE
       if (msg != "off") printf("printing ") > "/dev/stderr";
       if (head != "off"){
           printHeader("/dev/stdout", "WORDLIST=", label, fileno, hTotalSW);
       }
       for (i = 1; i <= wordno; i++){
           id = order[i];
           word = revlist[id];
           printLine("/dev/stdout", word, id, count, fileno, hTotalSW);
           if ((i % 1000 == 0) && (msg != "off")) printf(".") > "/dev/stderr";
       }
       if (total != "off"){
           printFooter("/dev/stdout", "TOTAL:", vTotal, fileno, hTotalSW);
       }
       if (msg != "off") printf(" done.\n") > "/dev/stderr";
# (21) PRINT PROPORTIONAL DATA FILE  (default N. Freq. setting is 10,000)
       if (prop != ""){
           if (msg != "off") printf("printing \"%s\" ", prop) > "/dev/stderr";
           if (head != "off"){
               printHeader(prop, "WORDLIST=", label, fileno, hTotalSW);
           }
           for (f = 1; f <= fileno; f++){
               vTotal[0] += vTotal[f];
           }
           for (i = 1; i <= wordno; i++){
              id = order[i];
              word = revlist[id];
              sum = 0;
              for (f = 1; f <= fileno; f++){
                   if ((f, id) in count){
                       if (vTotal[f] != 0){
                       data[f] = count[f, id] * 10000 / vTotal[f];
                       }else{
                          data[f] = 0;
                       }
                       sum += count[f, id];
                   }else{
                       data[f] = 0;
                   }

                   data[0] = sum * 10000 / vTotal[0];
               }
               printData(prop, word, data, fileno, hTotalSW);
              if ((i % 1000 == 0) && (msg != "off")) printf(".") > "/dev/stderr";
               }
              if (msg != "off") printf(" done.\n") > "/dev/stderr";
       }
# USE BASE FILE
   }else{
       if ((getline < base) <= 0){
           print "The specified base file", base, "cannot be read";
           exit;
       }
       if (msg != "off") printf("printing ") > "/dev/stderr";
       if (head != "off"){
           printHeader("/dev/stdout", "WORDLIST=", label, fileno, hTotalSW);
       }
       i = 0;
       do {
```

```
          word = $1;
          if (word in wordlist){
             id = wordlist[word];
             printLine("/dev/stdout", word, id, count, fileno, hTotalSW);
             for (f = 1; f <= fileno; f++){
                 if ((f, id) in count){
                     bTotal[f] += count[f, id];
                 }
             }
             if (misc != ""){
                mark[id] = id;
             }
          }else{
              printLine("/dev/stdout", word, 0, count, fileno, hTotalSW);
          }
         i++;
          if ((i % 1000 == 0) && (msg != "off")) printf(".") > "/dev/stderr";
       } while ((getline < base ) > 0);
       if (total != "off"){
           printFooter("/dev/stdout", "TOTAL:", bTotal, fileno, hTotalSW);
       }
       if (msg != "off") printf("done.\n") > "/dev/stderr";
# MISC. FILE
      if (misc != ""){
# CREATE A SORTED WORD LIST IN ORDER[]
         sz = 0;
         for (i = 1; i <= wordno; i++){
            if (!(i in mark)){ order[++sz] = i; }
         }
         for (word in wordlist){
            revlist[wordlist[word]] = word;
            delete wordlist[word];
         }
         if (msg != "off") printf("sorting %d words ", sz) > "/dev/stderr";
         qsort(order,revlist,1,sz);
         if (msg != "off") printf("done.\n") > "/dev/stderr";
# PRINT THE TABLE
         if (msg != "off") printf("printing \"%s\" ", misc) > "/dev/stderr";
         if (head != "off"){
            printHeader(misc, "WORDLIST=", label, fileno, hTotalSW);
         }
         for (i = 1; i <= sz; i++){
             id = order[i];
             word = revlist[id];
             printLine(misc, word, id, count, fileno, hTotalSW);
             for (f = 1; f <= fileno; f++){
                if ((f, id) in count) mTotal[f] += count[f, id];
             }
             if ((i % 1000 == 0) && (msg != "off")) printf(".") > "/dev/stderr";
           }
           if (total != "off"){
               printFooter(misc, "TOTAL:", mTotal, fileno, hTotalSW);
           }
           if (msg != "off") printf(" done.\n") > "/dev/stderr";
       }
   }
}
# End of Program
```