```
# An options parser for Thompson AWK
# (c) Markus Gnam 2017 Version 2017-06-05
# Notes:
#  You can use -, -- or / as first option character(s).
#  E.g. -diff=1, --diff=1, /diff=1 are all valid alternatives.
#  {0,1} means: Use "0" for No (False) or "1" for Yes (True).
#  For these {0,1} options omitting these values means Yes.
#  E.g. option -d, --diff={0,1}
#  You can use either --diff=1 or simply --diff
#  For the short option you can use -d, -d=1 or -d 1


function argparse(action, nargs, o_short, o_long, o_value, o_default, o_specific,
o_interfering) {
  # Actions: "create" or "add" or "make". Result: Global array OPTION_VALUES
  local debug = 0, file, indexopt, stringopt, lastopt, opt, i, j, key, value

  if (action == "create") {
    ### FROM CMD LINE, LOAD ARGS AND OPTIONS
    args[0] = 0
    for(arg=1; arg<ARGC; arg++) {
      if (ARGV[arg] ~ /^[-/]/) {
        indexopt = index(ARGV[arg], "=")
        if (indexopt == 0)
          stringopt = tolower(ARGV[arg])
        else
          stringopt = tolower(substr(ARGV[arg],1,indexopt)) substr(ARGV[arg],indexopt+1)
        OPTIONS[stringopt] = ARGV[arg]
      }
      else {
        # Last Short option (key) + argument (value), e.g. -f \t
        lastopt = ""
        if (arg-1 in ARGV)
          if ( (args[0] >= nargs) || ((args[0] < nargs) && (ARGC-arg > nargs)) )
            lastopt = tolower(ARGV[arg-1])
        if ( (lastopt != "") && (lastopt in OPTIONS) && (length(lastopt) == 2) ) {
          OPTIONS[lastopt "=" ARGV[arg]] = OPTIONS[lastopt] " " ARGV[arg]
          delete OPTIONS[lastopt]
        }
        else {
          # "Normal" arguments
          args[0]++
          args[args[0]] = ARGV[arg]
        }
      }
    }

    ### RESET ARGC AND ARGV FOR REAL ARGS
    for (i in ARGV) if (i > 0) delete ARGV[i]
    for(i=1; i <= args[0]; i++)
      ARGV[i] = args[i]
    ARGC = args[0] + 1

    ### TEST ARGS
    if (ARGC != nargs+1) {
      if (ARGC != 1) {
        print "\nWRONG USAGE: Too many or too few input files."
        usageprint()
        abort
```

```
    }
  }
  else {
    for(i=1; i<=ARGC-1; i++) {
      if (filemode(ARGV[i]) == "") {
        print "\nWRONG USAGE: Input file(s) not found."
        usageprint()
        abort
      }
      file = tolower(ARGV[i])
      if (file in FILES) {
        print "WARNING: File ARGV[" i "]=" ARGV[i] " is the same as ARGV[" FILES[file]
        "]=" ARGV[FILES[file]]
        print ""
      }
      else {
        FILES[file] = i
      }
    }
  }
}
else if (action == "add") {
  # Add option: Required (nargs=1), Option short, Option long, Value, Default, Specific,
  Interfering

  # Allowed keys
  # e.g. ALLOWEDKEYS["f"] = "--fs"
  # e.g. ALLOWEDKEYS["fs"] = "--fs"
  if (o_short && o_long) ALLOWEDKEYS[o_short] = "--" o_long
  if (o_long) ALLOWEDKEYS[o_long] = "--" o_long

  # Allowed values
  # e.g. ALLOWEDVALUES["f"] = regex(".+")
  # e.g. ALLOWEDVALUES["fs"] = regex(".+")
  if (o_short && o_value) ALLOWEDVALUES[o_short] = regex(o_value)
  if (o_long && o_value) ALLOWEDVALUES[o_long] = regex(o_value)

  # Default values
  # e.g. DEFAULTVALUES["--fs"] = " "
  if (o_long && o_default) DEFAULTVALUES["--" o_long] = o_default

  # Required values
  # e.g. REQUIREDVALUES["--out"] = ""
  if (o_long && nargs+0) REQUIREDVALUES["--" o_long] = o_default

  # Specific values
  # e.g. SPECIFICVALUES["--key1"] = "--key"
  # e.g. SPECIFICVALUES["--key2"] = "--key"
  if (o_long && o_specific) SPECIFICVALUES["--" o_long] = "--" o_specific

  # Interfering values
  # e.g. INTERFERINGVALUES["--trim"]["--ltrim=1"] = 1
  # e.g. INTERFERINGVALUES["--trim"]["--rtrim=1"] = 1
  if (o_long && o_interfering) {
    split(o_interfering, INTERFERINGARRAY, ",")
    for (i in INTERFERINGARRAY) {
      if (ALLOWEDVALUES[o_long] "" == "^[01]$")
        INTERFERINGVALUES["--" o_long]["--" INTERFERINGARRAY[i]] = 1
      else
```

```
        INTERFERINGVALUES["--" o_long]["--" INTERFERINGARRAY[i]] = 0
    }
  }
}
else if (action == "make") {
  ### TEST OPTIONS
  for (i in OPTIONS) {
    opt = i
    sub(/^[-/][-/]?/, "", opt)
    if ((opt in ALLOWEDVALUES) && (ALLOWEDVALUES[opt] "" == "^[01]$"))
      opt = opt "=1" # Set 1 for 0/1 values if not specified, e.g. set -ltrim to -ltrim=1
    key = substr(opt, 1, index(opt, "=") - 1)
    value  = substr(opt, index(opt, "=") + 1)
    if (debug) {
      print "opt: #" opt "#"
      print "key: #" key "#"
      print "value: #" value "#"
    }
    if ( (! (key in ALLOWEDKEYS)) || (value !~ ALLOWEDVALUES[key]) ) {
      print "\nWRONG USAGE: Option not allowed: " OPTIONS[i]
      usageprint()
      abort
    }
    else {
      # Search for duplicate keys
      if (ALLOWEDKEYS[key] in OPTION_VALUES) {
        print "\nWRONG USAGE: Duplicate option not allowed: " OPTIONS[i]
        usageprint()
        abort
      }
  ### MAKE OPTIONS
      # Save options (keys and values) neatly
      OPTION_VALUES[ALLOWEDKEYS[key]] = value
    }
  }
  ### SPECIAL OPTIONS
  # Get help
  if (ARGC == 1) {
    if (length(OPTION_VALUES) == 0) {
      usageprint()
    }
    else {
      if (("--help" in OPTION_VALUES) && (length(OPTION_VALUES) == 1)) {
        usageprint()
        optionsprint()
      }
      else {
        print "\nWRONG USAGE: Without arguments the only allowed option is getting help."
        usageprint()
      }
    }
    abort
  }
  else if ("--help" in OPTION_VALUES) {
    print "\nWRONG USAGE: Getting help not allowed with other arguments."
    usageprint()
    abort
  }
  # Specific values
```

```
for (i in OPTION_VALUES) {
  if ((i in SPECIFICVALUES) && (SPECIFICVALUES[i] in OPTION_VALUES)) {
    print "\nWRONG USAGE: General and specific option not allowed for " SPECIFICVALUES[i]
    usageprint()
    abort
  }
}
for (i in SPECIFICVALUES) {
  if (SPECIFICVALUES[i] in SPECIFIC_TARGET)
    ++SPECIFIC_TARGET[SPECIFICVALUES[i]]
  else
     SPECIFIC_TARGET[SPECIFICVALUES[i]] = 1
}
for (i in SPECIFICVALUES) {
  if (i in OPTION_VALUES) {
    if (SPECIFICVALUES[i] in SPECIFIC_ACTUAL)
      ++SPECIFIC_ACTUAL[SPECIFICVALUES[i]]
    else
       SPECIFIC_ACTUAL[SPECIFICVALUES[i]] = 1
  }
  else if (SPECIFICVALUES[i] in OPTION_VALUES) {
    OPTION_VALUES[i] = OPTION_VALUES[SPECIFICVALUES[i]]
  }
}
for (i in SPECIFIC_ACTUAL) {
  if (SPECIFIC_ACTUAL[i] != SPECIFIC_TARGET[i]) {
    print "\nWRONG USAGE: Too few specific options for " i
    usageprint()
    abort
  }
}
# Default values
for (i in DEFAULTVALUES) {
  if (! (i in OPTION_VALUES) ) {
    if (! (i in SPECIFIC_ACTUAL) ) {
      OPTION_VALUES[i] = DEFAULTVALUES[i]
      DEFAULT_ACTUAL[i] = i
    }
  }
  else {
    OPTIONS_ACTUAL[i] = i
  }
}
# Required values
for (i in REQUIREDVALUES) {
  if (! (i in OPTION_VALUES) ) {
    if (REQUIREDVALUES[i] == "") {
      print "\nWRONG USAGE: Required option " i " is missing"
      usageprint()
      abort
    }
  }
}
# Interfering values
for (i in INTERFERINGVALUES) {
  for (j in INTERFERINGVALUES[i]) {
    key = substr(j, 1, index(j, "=") - 1)
    value  = substr(j, index(j, "=") + 1)
    if ((key in OPTION_VALUES) && (key in DEFAULT_ACTUAL) && (INTERFERINGVALUES[i][j] ==
```

```
        "1")) {
          if (i in OPTIONS_ACTUAL) {
            if (OPTION_VALUES[key] == value) {
              if (OPTION_VALUES[key] == DEFAULTVALUES[key]) {
                if (DEFAULTVALUES[key] == "1")
                  OPTION_VALUES[key] = "0"
                else
                  OPTION_VALUES[key] = "1"
              }
            }
          }
        }
        if ((key in OPTION_VALUES) && (OPTION_VALUES[key] == value) && (i in OPTION_VALUES)) {
          if (((INTERFERINGVALUES[i][j] == "1") && (OPTION_VALUES[i] == "1")) \
           || ((INTERFERINGVALUES[i][j] == "0") && (!( i in DEFAULT_ACTUAL)))) {
            print "\nWRONG USAGE: Interfering options: " i "=" OPTION_VALUES[i] " and " j
            usageprint()
            abort
          }
        }
      }
    }

    ### DIAGNOSTIC: PRINT ARGV AND OPTIONS
    for(i=1; i<=ARGC-1; i++) print "ARGV[" i "]=" ARGV[i]
    printf("%s", "Options:")
    for (i in OPTION_VALUES) { printf(" %s=%s", i, OPTION_VALUES[i]) }
    printf("%s", "\n")
    if (length(OPTION_VALUES) == 0) print "[none]"

    ### ZAP UNNEEDED GLOBAL VARS
    delete args
    delete FILES
    delete ALLOWEDKEYS
    delete ALLOWEDOPTIONS
    delete DEFAULTVALUES
    delete DEFAULT_ACTUAL
    delete REQUIREDVALUES
    delete SPECIFICVALUES
    delete SPECIFIC_TARGET
    delete SPECIFIC_ACTUAL
    delete INTERFERINGVALUES
    delete INTERFERINGARRAY
    delete OPTIONS
    delete OPTIONS_ACTUAL
  }
}

function usageprint() {
  print ""
  print "listcompare.exe - Compare two lists based on a key"
  print "(c) Markus Gnam 2017. Version 1.0.0.0 20170602"
  print "USAGE: listcompare [options] <file1> <file2>"
  print ""
  print "Compare two files based on a key given for each file."
  print "They don't need to be sorted. There is no size limit."
  print "Defaults, if no options are specified:"
  print "Key for comparisons for both files is the first field."
  print "Default field separator is white space (spaces/tabs)."
```

```
  print ""
  print "Result output (if keys for these files exist):"
  print "key in both files => File: " key_both_in_file1_and_file2
  print "key only in file1 => File: " key_only_in_file1
  print "key only in file2 => File: " key_only_in_file2
  print ""
  print "For additional output of the value files use \"-v\""
  print "Value files contain the whole line where the key occurs."
  print "For all options and more details type \"listcompare --help\""
  print ""
}

function optionsprint() {
  print "OPTIONS: Details about all options and their defaults:"
  print "Mandatory arguments to long options are mandatory for short options too."
  print "  -f, --fs=FIELDSEP    The field separator [default: \" \"]"
  print "                       E.g. tabulator: --fs=\\t or one space: --fs=\"[ ]\""
  print "      --fs1=FIELDSEP   The field separator of file 1 [default: \" \"]"
  print "      --fs2=FIELDSEP   The field separator of file 2 [default: \" \"]"
  print "  -k, --key=FIELDNO    The field number to compare [default: 1]"
  print "                       Use --key=0 to get the whole line as key."
  print "                       For Special cases see Notes. E.g. part of field 8:"
  print "                       Field No[,Substr Pos][,Substr Length]: --key=8,1,4"
  print "                       A field name instead of a field number can be used:"
  print "                       E.g. --key=7 or with fname=: --key=fname=last_name"
  print "      --key1=FIELDNO   The field number of file 1 to compare [default: 1]"
  print "      --key2=FIELDNO   The field number of file 2 to compare [default: 1]"
  print "      --condition1=RE  Condition for key1 to be fulfilled [default: 0]"
  print "                       RE=<Field No>:~:<regex in double quotes>. See Notes."
  print "      --condition2=RE  Condition for key2 to be fulfilled [default: 0]"
  print "                       RE=<Field No>:~:<regex in double quotes>. See Notes."
  print "  -v, --value={0,1}    Generate files with values (whole lines)"
  print "                       additional to the key files [default: 0]"
  print "  -d, --diff={0,1}     Generate a difference file for values [default: 0]"
  print "      --dif1=FIELDNO   The field number of file 1 to compare [default: 0]"
  print "                       Use --dif1=0 for the whole line as value. See Notes."
  print "      --dif2=FIELDNO   The field number of file 2 to compare [default: 0]"
  print "                       Use --dif2=0 for the whole line as value. See Notes."
  print "  -h, --header=N       Take care of N header lines [default: 0]"
  print "      --header1=N      Take care of N header lines of file 1 [default: 0]"
  print "      --header2=N      Take care of N header lines of file 2 [default: 0]"
  print "      --trim={0,1}     Trim the key [default: 1]"
  print "      --ltrim={0,1}    Ltrim the key [default: 0]"
  print "      --rtrim={0,1}    Rtrim the key [default: 0]"
  print "      --upper={0,1}    Set the key to upper case [default: 0]"
  print "      --lower={0,1}    Set the key to lower case [default: 0]"
  print "      --lpadzero=N     Left pad the key with \"0\" to length N [default: 0]"
  print "      --csv1={0,1}     Dequote CSV file for file 1 [default: 0]"
  print "      --csv2={0,1}     Dequote CSV file for file 2 [default: 0]"
  print "      --utf81={0,1}    For substring use with UTF-8 file 1 [default: 0]"
  print "      --utf82={0,1}    For substring use with UTF-8 file 2 [default: 0]"
  print "      --debug={0,1}    Print debug files [default: 0]"
  print "      --largefiles=N   Split file into pieces of N lines [default: 1000000]"
  print "  -?, --help           Display this help and exit"
  print ""
  print "Notes:"
  print "  You can use -, -- or / as first option character(s)."
  print "  E.g. -diff=1, --diff=1, /diff=1 are all valid alternatives."
  print "  {0,1} means: Use \"0\" for No (False) or \"1\" for Yes (True)."
```

```
  print "  For these {0,1} options omitting these values means Yes."
  print "  E.g. option -d, --diff={0,1}"
  print "  You can use either --diff=1 or simply --diff"
  print "  For the short option you can use -d, -d=1 or -d 1"
  ...
}

BEGIN {
### Begin Options treatment:
  # Create argument parser
  # Two input files are allowed.
  argparse("create", 2)

  # Add options
  # Required?, Option short, Option long, Allowed values, Default value, Specific,
  Interfering.
  # "Required=1" means: There must be user input for this option and there is no Default
  value.
  # "Specific" means: Take care of the given interfering general option in case it is
  provided.
  # "Interfering" means a comma separated list of interfering options like "lower=1" for
  upper.
  argparse("add", "", "f", "fs", ".+", " ", "", "")
  argparse("add", "", "", "fs1", ".+", " ", "fs", "")
  argparse("add", "", "", "fs2", ".+", " ", "fs", "")
  argparse("add", "", "k", "key", "^[0-9]+|fname=", "1", "", "")
  argparse("add", "", "", "key1", "^[0-9]+|fname=", "1", "key", "")
  argparse("add", "", "", "key2", "^[0-9]+|fname=", "1", "key", "")
  argparse("add", "", "v", "value", "^[01]$", "0", "", "")
  argparse("add", "", "d", "diff", "^[01]$", "0", "", "")
  argparse("add", "", "", "dif1", "^[0-9]+|fname=", "0", "", "diff=0")
  argparse("add", "", "", "dif2", "^[0-9]+|fname=", "0", "", "diff=0")
  argparse("add", "", "h", "header", "^[0-9]+$", "0", "", "")
  argparse("add", "", "", "header1", "^[0-9]+$", "0", "header", "")
  argparse("add", "", "", "header2", "^[0-9]+$", "0", "header", "")
  argparse("add", "", "", "trim", "^[01]$", "1", "", "ltrim=1,rtrim=1")
  argparse("add", "", "", "ltrim", "^[01]$", "0", "", "trim=1")
  argparse("add", "", "", "rtrim", "^[01]$", "0", "", "trim=1")
  argparse("add", "", "", "upper", "^[01]$", "0", "", "lower=1")
  argparse("add", "", "", "lower", "^[01]$", "0", "", "upper=1")
  argparse("add", "", "", "lpadzero", "^[0-9]+", "0", "", "")
  argparse("add", "", "", "csv1", "^[01]$", "0", "", "")
  argparse("add", "", "", "csv2", "^[01]$", "0", "", "")
  argparse("add", "", "", "debug", "^[01]$", "0", "", "")
  argparse("add", "", "", "condition1", "^[0-9]+|fname=", "0", "", "")
  argparse("add", "", "", "condition2", "^[0-9]+|fname=", "0", "", "")
  argparse("add", "", "", "utf81", "^[01]$", "0", "", "")
  argparse("add", "", "", "utf82", "^[01]$", "0", "", "")
  argparse("add", "", "", "largefiles", "^[1-9][0-9]*$", "1000000", "", "")
  argparse("add", "", "?", "help", "^[01]$", "0", "", "")

  # Test and save options
  argparse("make")

  # Result: OPTION_VALUES array with keys and values,
  # e.g. OPTION_VALUES["--trim"] = "1"
### End Options treatment.

  # Preparation:
```

```
  ...

  # Use of large files support:
  arraynumber = OPTION_VALUES["--largefiles"] + 0

  # Read file 1:
  OFS = FS = OPTION_VALUES["--fs1"] # Set Field separator from options
  ...
}
```