```
**********************************************************
   16906 Sat Jun 13 17:15:03 2015
new/usr/src/cmd/spell/spellprog.c
3727 british people can’t spell
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <locale.h>
#include "hash.h"

#define Tolower(c) (isupper(c)?tolower(c):c)
#define DLEV 2

/*
 * ANSI prototypes
 */
static int      ily(char *, char *, char *, int);
static int      s(char *, char *, char *, int);
static int      es(char *, char *, char *, int);
static int      subst(char *, char *, char *, int);
static int      nop(void);
static int      bility(char *, char *, char *, int);
static int      i_to_y(char *, char *, char *, int);
static int      CCe(char *, char *, char *, int);
static int      y_to_e(char *, char *, char *, int);
static int      strip(char *, char *, char *, int);
static int      ize(char *, char *, char *, int);
static int      tion(char *, char *, char *, int);
static int      an(char *, char *, char *, int);
int             prime(char *);
```

```
static void     ise(void);
static int      tryword(char *, char *, int);
static int      trypref(char *, char *, int);
static int      trysuff(char *, int);
static int      vowel(int);
static int      dict(char *, char *);
static int      monosyl(char *, char *);
static int      VCe(char *, char *, char *, int);
static char     *skipv(char *);
static void     ztos(char *);

struct suftab {
static struct suftab {
        char *suf;
        int (*p1)();
        int n1;
        char *d1;
        char *a1;
        int (*p2)();
        int n2;
        char *d2;
        char *a2;
};

static struct suftab sufa[] = {
} suftab[] = {
        {"ssen", ily, 4, "-y+iness", "+ness" },
        {"ssel", ily, 4, "-y+i+less", "+less" },
        {"se", s, 1, "", "+s",  es, 2, "-y+ies", "+es" },
        {"s’", s, 2, "", "+’s"},
        {"s", s, 1, "", "+s"},
        {"ecn", subst, 1, "-t+ce", ""},
        {"ycn", subst, 1, "-t+cy", ""},
        {"ytilb", nop, 0, "", ""},
        {"ytilib", bility, 5, "-le+ility", ""},
        {"elbaif", i_to_y, 4, "-y+iable", ""},
        {"elba", CCe, 4, "-e+able", "+able"},
        {"yti", CCe, 3, "-e+ity", "+ity"},
        {"ylb", y_to_e, 1, "-e+y", ""},
        {"yl", ily, 2, "-y+ily", "+ly"},
        {"laci", strip, 2, "", "+al"},
        {"latnem", strip, 2, "", "+al"},
        {"lanoi", strip, 2, "", "+al"},
        {"tnem", strip, 4, "", "+ment"},
        {"gni", CCe, 3, "-e+ing", "+ing"},
        {"reta", nop, 0, "", ""},
        {"retc", nop, 0, "", ""},
        {"re", strip, 1, "", "+r", i_to_y, 2, "-y+ier", "+er"},
        {"de", strip, 1, "", "+d", i_to_y, 2, "-y+ied", "+ed"},
        {"citsi", strip, 2, "", "+ic"},
        {"citi", ize, 1, "-ic+e", ""},
        {"cihparg", i_to_y, 1, "-y+ic", ""},
        {"tse", strip, 2, "", "+st",    i_to_y, 3, "-y+iest", "+est"},
        {"cirtem", i_to_y, 1, "-y+ic", ""},
        {"yrtem", subst, 0, "-er+ry", ""},
        {"cigol", i_to_y, 1, "-y+ic", ""},
        {"tsigol", i_to_y, 2, "-y+ist", ""},
        {"tsi", CCe, 3, "-e+ist", "+ist"},
        {"msi", CCe, 3, "-e+ism", "+ist"},
        {"noitacifi", i_to_y, 6, "-y+ication", ""},
        {"noitazi", ize, 4, "-e+ation", ""},
        {"rota", tion, 2, "-e+or", ""},
        {"rotc", tion, 2, "", "+or"},
        {"noit", tion, 3, "-e+ion", "+ion"},
        {"naino", an, 3, "", "+ian"},
        {"na", an, 1, "", "+n"},
        {"evi", subst, 0, "-ion+ive", ""},
        {"ezi", CCe, 3, "-e+ize", "+ize"},
        {"pihs", strip, 4, "", "+ship"},
        {"dooh", ily, 4, "-y+ihood", "+hood"},
        {"luf", ily, 3, "-y+iful", "+ful"},
        {"ekil", strip, 4, "", "+like"},
        0
};

static struct suftab sufb[] = {
        {"ssen", ily, 4, "-y+iness", "+ness" },
        {"ssel", ily, 4, "-y+i+less", "+less" },
        {"se", s, 1, "", "+s",  es, 2, "-y+ies", "+es" },
        {"s'", s, 2, "", "+'s"},
        {"s", s, 1, "", "+s"},
        {"ecn", subst, 1, "-t+ce", ""},
        {"ycn", subst, 1, "-t+cy", ""},
        {"ytilb", nop, 0, "", ""},
        {"ytilib", bility, 5, "-le+ility", ""},
        {"elbaif", i_to_y, 4, "-y+iable", ""},
        {"elba", CCe, 4, "-e+able", "+able"},
        {"yti", CCe, 3, "-e+ity", "+ity"},
        {"ylb", y_to_e, 1, "-e+y", ""},
        {"yl", ily, 2, "-y+ily", "+ly"},
        {"laci", strip, 2, "", "+al"},
        {"latnem", strip, 2, "", "+al"},
        {"lanoi", strip, 2, "", "+al"},
        {"tnem", strip, 4, "", "+ment"},
        {"gni", CCe, 3, "-e+ing", "+ing"},
        {"reta", nop, 0, "", ""},
        {"retc", nop, 0, "", ""},
        {"re", strip, 1, "", "+r", i_to_y, 2, "-y+ier", "+er"},
        {"de", strip, 1, "", "+d", i_to_y, 2, "-y+ied", "+ed"},
        {"citsi", strip, 2, "", "+ic"},
        {"citi", ize, 1, "-ic+e", ""},
        {"cihparg", i_to_y, 1, "-y+ic", ""},
        {"tse", strip, 2, "", "+st",    i_to_y, 3, "-y+iest", "+est"},
        {"cirtem", i_to_y, 1, "-y+ic", ""},
        {"yrtem", subst, 0, "-er+ry", ""},
        {"cigol", i_to_y, 1, "-y+ic", ""},
        {"tsigol", i_to_y, 2, "-y+ist", ""},
        {"tsi", CCe, 3, "-e+ist", "+ist"},
        {"msi", CCe, 3, "-e+ism", "+ist"},
        {"noitacifi", i_to_y, 6, "-y+ication", ""},
        {"noitasi", ize, 4, "-e+ation", ""},
        {"rota", tion, 2, "-e+or", ""},
        {"rotc", tion, 2, "", "+or"},
        {"noit", tion, 3, "-e+ion", "+ion"},
        {"naino", an, 3, "", "+ian"},
        {"na", an, 1, "", "+n"},
        {"evi", subst, 0, "-ion+ive", ""},
        {"esi", CCe, 3, "-e+ise", "+ise"},
        {"pihs", strip, 4, "", "+ship"},
        {"dooh", ily, 4, "-y+ihood", "+hood"},
        {"luf", ily, 3, "-y+iful", "+ful"},
        {"ekil", strip, 4, "", "+like"},
        0
};

static char *preftab[] = {
        "anti",
        "auto",
        "bio",
        "counter",
        "dis",
        "electro",
```

```
        "en",
        "fore",
        "geo",
        "hyper",
        "intra",
        "inter",
        "iso",
        "kilo",
        "magneto",
        "meta",
        "micro",
        "mid",
        "milli",
        "mis",
        "mono",
        "multi",
        "non",
        "out",
        "over",
        "photo",
        "poly",
        "pre",
        "pseudo",
        "psycho",
        "re",
        "semi",
        "stereo",
        "sub",
        "super",
        "tele",
        "thermo",
        "ultra",
        "under",        /* must precede un */
        "un",
        0
};

static int bflag;
static int vflag;
static int xflag;
static struct suftab *suftab;
static char *prog;
static char word[LINE_MAX];
static char original[LINE_MAX];
static char *deriv[LINE_MAX];
static char affix[LINE_MAX];
static FILE *file, *found;
/*
 *      deriv is stack of pointers to notes like +micro +ed
 *      affix is concatenated string of notes
 *      the buffer size 141 stems from the sizes of original and affix.
 */

/*
 *      in an attempt to defray future maintenance misunderstandings, here is
 *      an attempt to describe the input/output expectations of the spell
 *      program.
 *
 *      spellprog is intended to be called from the shell file spell.
 *      because of this, there is little error checking (this is historical, not
 *      necessarily advisable).
 *
 *      spellprog options hashed-list pass
 *
 *      the hashed-list is a list of the form made by spellin.
 *      there are 2 types of hashed lists:
```

```
 *             1. a stop list: this specifies words that by the rules embodied
 *                in spellprog would be recognized as correct, BUT are really
 *                errors.
 *             2. a dictionary of correctly spelled words.
 *      the pass number determines how the words found in the specified
 *      hashed-list are treated. If the pass number is 1, the hashed-list is
 *      treated as the stop-list, otherwise, it is treated as the regular
 *      dictionary list. in this case, the value of "pass" is a filename. Found
 *      words are written to this file.
 *
 *      In the normal case, the filename = /dev/null. However, if the v option
 *      is specified, the derivations are written to this file.
 *      The spellprog looks up words in the hashed-list; if a word is found, it
 *      is printed to the stdout. If the hashed-list was the stop-list, the
 *      words found are presumed to be misspellings. in this case,
 *      a control character is printed ( a "-" is appended to the word.
 *      a hyphen will never occur naturally in the input list because deroff
 *      is used in the shell file before calling spellprog.)
 *      If the regualar spelling list was used (hlista or hlistb), the words
 *      are correct, and may be ditched. (unless the -v option was used -
 *      see the manual page).
 *
 *      spellprog should be called twice : first with the stop-list, to flag all
 *      a priori incorrectly spelled words; second with the dictionary.
 *
 *      spellprog hstop 1 |\
 *      spellprog hlista /dev/null
 *
 *      for a complete scenario, see the shell file: spell.
 *
 */

int
main(int argc, char **argv)
{
        char *ep, *cp;
        char *dp;
        int fold;
        int c, j;
        int pass;

        /* Set locale environment variables local definitions */
        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it wasn't */
#endif
        (void) textdomain(TEXT_DOMAIN);


        prog = argv[0];
        while ((c = getopt(argc, argv, "bvx")) != EOF) {
                switch (c) {
                case 'b':
                        bflag++;
                        ise();
                        break;
                case 'v':
                        vflag++;
                        break;
                case 'x':
                        xflag++;
                        break;
                }
        }

        argc -= optind;
```

```
        argv = &argv[optind];

        if ((argc < 2) || !prime(*argv)) {
                (void) fprintf(stderr,
                    gettext("%s: cannot initialize hash table\n"), prog);
                exit(1);
        }
        argc--;
        argv++;

        /* Select the correct suffix table */
        suftab = (bflag == 0) ? sufa : sufb;

/*
 *      if pass is not 1, it is assumed to be a filename.
 *      found words are written to this file.
 */
        pass = **argv;
        if (pass != '1')
                found = fopen(*argv, "w");

        for (;;) {
                affix[0] = 0;
                file = stdout;
                for (ep = word; (*ep = j = getchar()) != '\n'; ep++)
                        if (j == EOF)
                                exit(0);
/*
 *      here is the hyphen processing. these words were found in the stop
 *      list. however, if they exist as is, (no derivations tried) in the
 *      dictionary, let them through as correct.
 *
 */
                if (ep[-1] == '-') {
                        *--ep = 0;
                        if (!tryword(word, ep, 0))
                                (void) fprintf(file, "%s\n", word);
                        continue;
                }
                for (cp = word, dp = original; cp < ep; )
                        *dp++ = *cp++;
                *dp = 0;
                fold = 0;
                for (cp = word; cp < ep; cp++)
                        if (islower(*cp))
                                goto lcase;
                if (((ep - word) == 1) &&
                    ((word[0] == 'A') || (word[0] == 'I')))
                        continue;
                if (trypref(ep, ".", 0))
                        goto foundit;
                ++fold;
                for (cp = original+1, dp = word+1; dp < ep; dp++, cp++)
                        *dp = Tolower(*cp);
lcase:
                if (((ep - word) == 1) && (word[0] == 'a'))
                        continue;
                if (trypref(ep, ".", 0)||trysuff(ep, 0))
                        goto foundit;
                if (isupper(word[0])) {
                        for (cp = original, dp = word; *dp = *cp++; dp++)
                                if (fold) *dp = Tolower(*dp);
                        word[0] = Tolower(word[0]);
                        goto lcase;
                }
                (void) fprintf(file, "%s\n", original);
                continue;

foundit:
                if (pass == '1')
                        (void) fprintf(file, "%s-\n", original);
                else if (affix[0] != 0 && affix[0] != '.') {
                        file = found;
                        (void) fprintf(file, "%s\t%s\n", affix,
                            original);
                }
        }
}

/*
 *      strip exactly one suffix and do
 *      indicated routine(s), which may recursively
 *      strip suffixes
 */

static int
trysuff(char *ep, int lev)
{
        struct suftab   *t;
        char *cp, *sp;

        lev += DLEV;
        deriv[lev] = deriv[lev-1] = 0;
        for (t = &suftab[0]; (t != 0 && (sp = t->suf) != 0); t++) {
        for (t = &suftab[0]; (sp = t->suf) != 0; t++) {
                cp = ep;
                while (*sp)
                        if (*--cp != *sp++)
                                goto next;
                for (sp = cp; --sp >= word && !vowel(*sp); )
                        ;
                for (sp = cp; --sp >= word && !vowel(*sp); );
                if (sp < word)
                        return (0);
                if ((*t->p1)(ep-t->n1, t->d1, t->a1, lev+1))
                        return (1);
                if (t->p2 != 0) {
                        deriv[lev] = deriv[lev+1] = 0;
                        return ((*t->p2)(ep-t->n2, t->d2, t->a2, lev));
                }
                return (0);
next:;
        }
        return (0);
}
```

______***unchanged_portion_omitted_***

```
/* crummy way to Britishise */
static void
ise(void)
{
        struct suftab *p;

        for (p = suftab; p->suf; p++) {
                ztos(p->suf);
                ztos(p->d1);
                ztos(p->a1);
        }
}

static void
ztos(char *s)
```

```
{
        for (; *s; s++)
                if (*s == 'z')
                        *s = 's';
}

static int
dict(char *bp, char *ep)
{
        int temp, result;
        if (xflag)
                (void) fprintf(stdout, "=%.*s\n", ep-bp, bp);
        temp = *ep;
        *ep = 0;
        result = hashlook(bp);
        *ep = temp;
        return (result);
}
```

______***unchanged_portion_omitted_***