```
**********************************************************
   12863 Thu May  8 09:00:23 2014
new/usr/src/cmd/printf/printf.c
4854 printf(1) doesn't support %b and \c properly
**********************************************************
```

```
/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 * Copyright 2010 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <sys/types.h>

#include <err.h>
#include <errno.h>
#include <inttypes.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <alloca.h>
#include <ctype.h>
#include <locale.h>
#include <note.h>

#define warnx1(a, b, c)         warnx(a)
#define warnx2(a, b, c)         warnx(a, b)
#define warnx3(a, b, c)         warnx(a, b, c)

#define PTRDIFF(x, y)   ((uintptr_t)(x) - (uintptr_t)(y))

#define _(x)    gettext(x)

#define PF(f, func) do {                                                \
        char *b = NULL;                                                 \
        if (havewidth)                                                  \
                if (haveprec)                                           \
                        (void) asprintf(&b, f, fieldwidth, precision, func); \
                else                                                    \
                        (void) asprintf(&b, f, fieldwidth, func);       \
        else if (haveprec)                                              \
                (void) asprintf(&b, f, precision, func);                \
        else                                                            \
                (void) asprintf(&b, f, func);                           \
        if (b) {                                                        \
                (void) fputs(b, stdout);                                \
                free(b);                                                \
        }                                                               \
_NOTE(CONSTCOND) } while (0)

static int       asciicode(void);
static char     *doformat(char *, int *);
static int       escape(char *, int, size_t *);
static int       getchr(void);
static int       getfloating(long double *, int);
static int       getint(int *);
static int       getnum(intmax_t *, uintmax_t *, int);
static const char
                *getstr(void);
static char     *mknum(char *, char);
static void      usage(void);

static const char digits[] = "0123456789";

static int  myargc;
static char **myargv;
static char **gargv;
static char **maxargv;

int
main(int argc, char *argv[])
{
        size_t len;
        int end, rval;
        int chopped, end, rval;
        char *format, *fmt, *start;

        (void) setlocale(LC_ALL, "");

        argv++;
        argc--;

        /*
         * POSIX says: Standard utilities that do not accept options,
         * but that do accept operands, shall recognize "--" as a
         * first argument to be discarded.
         */
        if (argc && strcmp(argv[0], "--") == 0) {
                argc--;
                argv++;
        }

        if (argc < 1) {
                usage();
                return (1);
        }

        /*
         * Basic algorithm is to scan the format string for conversion
         * specifications -- once one is found, find out if the field
         * width or precision is a '*'; if it is, gather up value.  Note,
         * format strings are reused as necessary to use up the provided
         * arguments, arguments of zero/null string are provided to use
         * up the format string.
         */
        fmt = format = *argv;
```

```
127         (void) escape(fmt, 1, &len);    /* backslash interpretation */
127         chopped = escape(fmt, 1, &len);         /* backslash interpretation */
128         rval = end = 0;
129         gargv = ++argv;

131         for (;;) {
132                 maxargv = gargv;

134                 myargv = gargv;
135                 for (myargc = 0; gargv[myargc]; myargc++)
136                         /* nop */;
137                 start = fmt;
138                 while (fmt < format + len) {
139                         if (fmt[0] == '%') {
140                                 (void) fwrite(start, 1, PTRDIFF(fmt, start),
141                                     stdout);
142                                 if (fmt[1] == '%') {
143                                         /* %% prints a % */
144                                         (void) putchar('%');
145                                         fmt += 2;
146                                 } else {
147                                         fmt = doformat(fmt, &rval);
148                                         if (fmt == NULL)
149                                                 return (1);
150                                         end = 0;
151                                 }
152                                 start = fmt;
153                         } else
154                                 fmt++;
155                         if (gargv > maxargv)
156                                 maxargv = gargv;
157                 }
158                 gargv = maxargv;

160                 if (end == 1) {
161                         warnx1(_("missing format character"), NULL, NULL);
162                         return (1);
163                 }
164                 (void) fwrite(start, 1, PTRDIFF(fmt, start), stdout);
165                 if (!*gargv)
165                 if (chopped || !*gargv)
166                         return (rval);
167                 /* Restart at the beginning of the format string. */
168                 fmt = format;
169                 end = 1;
170         }
171         /* NOTREACHED */
172 }


175 static char *
176 doformat(char *fmt, int *rval)
177 {
178         static const char skip1[] = "#'-+ 0";
179         int fieldwidth, haveprec, havewidth, mod_ldbl, precision;
180         char convch, nextch;
181         char *start;
182         char **fargv;
183         char *dptr;
184         int l;

186         start = alloca(strlen(fmt) + 1);

188         dptr = start;
189         *dptr++ = '%';
190         *dptr = 0;
```



```
192         fmt++;

194         /* look for "n$" field index specifier */
195         l = strspn(fmt, digits);
196         if ((l > 0) && (fmt[l] == '$')) {
197                 int idx = atoi(fmt);
198                 if (idx <= myargc) {
199                         gargv = &myargv[idx - 1];
200                 } else {
201                         gargv = &myargv[myargc];
202                 }
203                 if (gargv > maxargv) {
204                         maxargv = gargv;
205                 }
206                 fmt += l + 1;

208                 /* save format argument */
209                 fargv = gargv;
210         } else {
211                 fargv = NULL;
212         }

214         /* skip to field width */
215         while (strchr(skip1, *fmt) != NULL) {
216                 *dptr++ = *fmt++;
217                 *dptr = 0;
218         }


221         if (*fmt == '*') {

223                 fmt++;
224                 l = strspn(fmt, digits);
225                 if ((l > 0) && (fmt[l] == '$')) {
226                         int idx = atoi(fmt);
227                         if (idx <= myargc) {
228                                 gargv = &myargv[idx - 1];
229                         } else {
230                                 gargv = &myargv[myargc];
231                         }
232                         fmt += l + 1;
233                 }

235                 if (getint(&fieldwidth))
236                         return (NULL);
237                 if (gargv > maxargv) {
238                         maxargv = gargv;
239                 }
240                 havewidth = 1;

242                 *dptr++ = '*';
243                 *dptr = 0;
244         } else {
245                 havewidth = 0;

247                 /* skip to possible '.', get following precision */
248                 while (isdigit(*fmt)) {
249                         *dptr++ = *fmt++;
250                         *dptr = 0;
251                 }
252         }

254         if (*fmt == '.') {
255                 /* precision present? */
256                 fmt++;
```

```
257                 *dptr++ = '.';

259                 if (*fmt == '*') {

261                         fmt++;
262                         l = strspn(fmt, digits);
263                         if ((l > 0) && (fmt[l] == '$')) {
264                                 int idx = atoi(fmt);
265                                 if (idx <= myargc) {
266                                         gargv = &myargv[idx - 1];
267                                 } else {
268                                         gargv = &myargv[myargc];
269                                 }
270                                 fmt += l + 1;
271                         }

273                         if (getint(&precision))
274                                 return (NULL);
275                         if (gargv > maxargv) {
276                                 maxargv = gargv;
277                         }
278                         haveprec = 1;
279                         *dptr++ = '*';
280                         *dptr = 0;
281                 } else {
282                         haveprec = 0;

284                         /* skip to conversion char */
285                         while (isdigit(*fmt)) {
286                                 *dptr++ = *fmt++;
287                                 *dptr = 0;
288                         }
289                 }
290         } else
291                 haveprec = 0;
292         if (!*fmt) {
293                 warnx1(_("missing format character"), NULL, NULL);
294                 return (NULL);
295         }
296         *dptr++ = *fmt;
297         *dptr = 0;

299         /*
300          * Look for a length modifier.  POSIX doesn't have these, so
301          * we only support them for floating-point conversions, which
302          * are extensions.  This is useful because the L modifier can
303          * be used to gain extra range and precision, while omitting
304          * it is more likely to produce consistent results on different
305          * architectures.  This is not so important for integers
306          * because overflow is the only bad thing that can happen to
307          * them, but consider the command  printf %a 1.1
308          */
309         if (*fmt == 'L') {
310                 mod_ldbl = 1;
311                 fmt++;
312                 if (!strchr("aAeEfFgG", *fmt)) {
313                         warnx2(_("bad modifier L for %%%c"), *fmt, NULL);
314                         return (NULL);
315                 }
316         } else {
317                 mod_ldbl = 0;
318         }

320         /* save the current arg offset, and set to the format arg */
321         if (fargv != NULL) {
322                 gargv = fargv;
```



```
323         }

325         convch = *fmt;
326         nextch = *++fmt;

328         *fmt = '\0';
329         switch (convch) {
330         case 'b': {
331                 size_t len;
332                 char *p;
333                 int getout;

335                 p = strdup(getstr());
336                 if (p == NULL) {
337                         warnx2("%s", strerror(ENOMEM), NULL);
338                         return (NULL);
339                 }
340                 getout = escape(p, 0, &len);
341                 (void) fputs(p, stdout);
341                 *(fmt - 1) = 's';
342                 PF(start, p);
343                 *(fmt - 1) = 'b';
342                 free(p);

344                 if (getout)
345                         exit(*rval);
347                         return (fmt);
346                 break;
347         }
348         case 'c': {
349                 char p;

351                 p = getchr();
352                 PF(start, p);
353                 break;
354         }
355         case 's': {
356                 const char *p;

358                 p = getstr();
359                 PF(start, p);
360                 break;
361         }
362         case 'd': case 'i': case 'o': case 'u': case 'x': case 'X': {
363                 char *f;
364                 intmax_t val;
365                 uintmax_t uval;
366                 int signedconv;

368                 signedconv = (convch == 'd' || convch == 'i');
369                 if ((f = mknum(start, convch)) == NULL)
370                         return (NULL);
371                 if (getnum(&val, &uval, signedconv))
372                         *rval = 1;
373                 if (signedconv)
374                         PF(f, val);
375                 else
376                         PF(f, uval);
377                 break;
378         }
379         case 'e': case 'E':
380         case 'f': case 'F':
381         case 'g': case 'G':
382         case 'a': case 'A': {
383                 long double p;
```

```
385                 if (getfloating(&p, mod_ldbl))
386                         *rval = 1;
387                 if (mod_ldbl)
388                         PF(start, p);
389                 else
390                         PF(start, (double)p);
391                 break;
392         }
393         default:
394                 warnx2(_("illegal format character %c"), convch, NULL);
395                 return (NULL);
396         }
397         *fmt = nextch;

399         /* return the gargv to the next element */
400         return (fmt);
401 }
```

*unchanged_portion_omitted_*

```
429 static int
430 escape(char *fmt, int percent, size_t *len)
431 {
432         char *save, *store, c;
433         int value;

435         for (save = store = fmt; ((c = *fmt) != 0); ++fmt, ++store) {
436                 if (c != '\\') {
437                         *store = c;
438                         continue;
439                 }
440                 switch (*++fmt) {
441                 case '\0':              /* EOS, user error */
442                         *store = '\\';
443                         *++store = '\0';
444                         *len = PTRDIFF(store, save);
445                         return (0);
446                 case '\\':              /* backslash */
447                 case '\'':              /* single quote */
448                         *store = *fmt;
449                         break;
450                 case 'a':               /* bell/alert */
451                         *store = '\a';
452                         break;
453                 case 'b':               /* backspace */
454                         *store = '\b';
455                         break;
456                 case 'c':
457                         if (!percent) {
458                                 *store = '\0';
459                                 *len = PTRDIFF(store, save);
460                                 return (1);
461                         }
462                         *store = 'c';
463                         break;
464                 case 'f':               /* form-feed */
465                         *store = '\f';
466                         break;
467                 case 'n':               /* newline */
468                         *store = '\n';
469                         break;
470                 case 'r':               /* carriage-return */
471                         *store = '\r';
472                         break;
473                 case 't':               /* horizontal tab */
474                         *store = '\t';
475                         break;
```



```
476                 case 'v':               /* vertical tab */
477                         *store = '\v';
478                         break;
479                                         /* octal constant */
480                 case '0': case '1': case '2': case '3':
481                 case '4': case '5': case '6': case '7':
482                         c = (!percent && *fmt == '0') ? 4 : 3;
483                         for (value = 0;
484                             c-- && *fmt >= '0' && *fmt <= '7'; ++fmt) {
485                                 value <<= 3;
486                                 value += *fmt - '0';
487                         }
488                         --fmt;
489                         if (percent && value == '%') {
490                                 *store++ = '%';
491                                 *store = '%';
492                         } else
493                                 *store = (char)value;
494                         break;
495                 default:
496                         *store = *fmt;
497                         break;
498                 }
499         }
500         *store = '\0';
501         *len = PTRDIFF(store, save);
502         return (0);
503 }
```

*unchanged_portion_omitted_*

```
***********************************************************
     613 Thu May  8 09:00:23 2014
new/usr/src/test/Makefile
4854 printf(1) doesn't support %b and \c properly
***********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  15 #

  17 .PARALLEL: $(SUBDIRS)

  19 SUBDIRS = os-tests test-runner util-tests zfs-tests
  18 SUBDIRS = os-tests test-runner zfs-tests

  21 include Makefile.com
```

```
**********************************************************
    4477 Thu May  8 09:00:23 2014
new/usr/src/test/util-tests/tests/printf/printf_test.ksh
4854 printf(1) doesn't support %b and \c properly
**********************************************************
______unchanged_portion_omitted_

  28 test_fail() {
  29         print "TEST FAIL: ${1}: ${2}"
  30         exit -1
  30 #        exit -1
  31 }
______unchanged_portion_omitted_

  45 typeset -A tests=()


  48 typeset -A tests[01]=()
  49 tests[01][desc]="hexadecimal lowercase"
  50 tests[01][format]='%04x'
  51 tests[01][args]="255"
  52 tests[01][result]="00ff"

  54 typeset -A tests[02]=()
  55 tests[02][desc]="hexadecimal 32-bit"
  56 tests[02][format]='%08x'
  57 tests[02][args]='65537'
  58 tests[02][result]=00010001

  60 typeset -A tests[03]=()
  61 tests[03][desc]="multiple arguments"
  62 tests[03][format]='%d %s '
  63 tests[03][args]="1 one 2 two 3 three"
  64 tests[03][result]='1 one 2 two 3 three '

  66 typeset -A tests[04]=()
  67 tests[04][desc]="variable position parameters"
  68 tests[04][format]='%2$s %1$d '
  69 tests[04][args]="1 one 2 two 3 three"
  70 tests[04][result]='one 1 two 2 three 3 '

  72 typeset -A tests[05]=()
  73 tests[05][desc]="width"
  74 tests[05][format]='%10s'
  75 tests[05][args]="abcdef"
  76 tests[05][result]='    abcdef'

  78 typeset -A tests[06]=()
  79 tests[06][desc]="width and precision"
  80 tests[06][format]='%10.3s'
  81 tests[06][args]="abcdef"
  82 tests[06][result]='       abc'

  84 typeset -A tests[07]=()
  85 tests[07][desc]="variable width and precision"
  86 tests[07][format]='%*.*s'
  87 tests[07][args]="10 3 abcdef"
  88 tests[07][result]='       abc'

  90 typeset -A tests[08]=()
  91 tests[08][desc]="variable position width and precision"
  92 tests[08][format]='%2$*1$.*3$s'
  93 tests[08][args]="10 abcdef 3"
  94 tests[08][result]='       abc'

  96 typeset -A tests[09]=()
```



```
  97 tests[09][desc]="multi variable position width and precision"
  98 tests[09][format]='%2$*1$.*3$s'
  99 tests[09][args]="10 abcdef 3 5 xyz 1"
 100 tests[09][result]='       abc    x'

 102 typeset -A tests[10]=()
 103 tests[10][desc]="decimal from hex"
 104 tests[10][format]='%d '
 105 tests[10][args]="0x1000 0XA"
 106 tests[10][result]='4096 10 '

 108 typeset -A tests[11]=()
 109 tests[11][desc]="negative dec (64-bit)"
 110 tests[11][format]='%x'
 111 tests[11][args]="-1"
 112 tests[11][result]='ffffffffffffffff'

 114 typeset -A tests[12]=()
 115 tests[12][desc]="float (basic)"
 116 tests[12][format]='%f'
 117 tests[12][args]="3.14"
 118 tests[12][result]='3.140000'

 120 typeset -A tests[12]=()
 121 tests[12][desc]="float precision"
 122 tests[12][format]='%.2f'
 123 tests[12][args]="3.14159"
 124 tests[12][result]='3.14'

 126 typeset -A tests[13]=()
 127 tests[13][desc]="left justify"
 128 tests[13][format]='%-5d'
 129 tests[13][args]="45"
 130 tests[13][result]='45   '

 132 typeset -A tests[14]=()
 133 tests[14][desc]="newlines"
 134 tests[14][format]='%s\n%s\n%s'
 135 tests[14][args]="one two three"
 136 tests[14][result]='one
 137 two
 138 three'

 140 typeset -A tests[15]=()
 141 tests[15][desc]="embedded octal escape"
 142 tests[15][format]='%s\41%s'
 143 tests[15][args]="one two"
 144 tests[15][result]='one!two'

 146 typeset -A tests[16]=()
 147 tests[16][desc]="backslash string (%b)"
 148 tests[16][format]='%b'
 149 tests[16][args]='\0101\0102\0103'
 150 tests[16][result]='ABC'
 146 # this is not yet supported
 147 #typeset -A tests[16]=()
 148 #tests[16][desc]="backslash string (%b)"
 149 #tests[16][format]='%b'
 150 #tests[16][args]='\0101\0102\0103'
 151 #tests[16][result]='ABC'

 152 typeset -A tests[17]=()
 153 tests[17][desc]="backslash c in %b"
 154 tests[17][format]='%b%s'
 155 tests[17][args]='\0101\cone two'
 156 tests[17][result]='A'
```

```
153 # nor is this
154 #typeset -A tests[17]=()
155 #tests[17][desc]="backslash c in %b"
156 #tests[17][format]='%b%s'
157 #tests[17][args]='\0101\cone two'
158 #tests[17][result]='A'
```

```
158 typeset -A tests[18]=()
159 tests[18][desc]="backslash octal in format"
160 tests[18][format]='HI\1120K\0112tabbed\11again'
161 tests[18][args]=
162 tests[18][result]='HIJ0K        2tabbed again'

164 typeset -A tests[19]=()
165 tests[19][desc]="backslash octal in %b"
166 tests[19][format]="%b"
167 tests[19][args]='HI\0112K\011tabbed'
168 tests[19][result]='HIJK tabbed'

170 typeset -A tests[20]=()
171 tests[20][desc]="numeric %d and ASCII conversions"
172 tests[20][format]='%d '
173 tests[20][args]="3 +3 -3 \"3 \"+ '-"
174 tests[20][result]='3 3 -3 51 43 45 '
```

```
176 #debug=yes

178 for i in "${!tests[@]}"; do
179         t=test_$i
180         desc=${tests[$i][desc]}
181         format=${tests[$i][format]}
182         args="${tests[$i][args]}"
183         result=${tests[$i][result]}
184
185         test_start $t "${tests[$i][desc]}"
186         [[ -n "$debug" ]] && echo $PRINTF "$format" "${args[@]}"
187         comp=$($PRINTF "$format" ${args[@]})
188         checkrv $t
189         [[ -n "$debug" ]] && echo "got [$comp]"
190         good=$result
191         compare $t "$comp" "$good"
192         test_pass $t
193 done
```