```
**********************************************************
   12669 Fri May  2 20:33:49 2014
new/usr/src/cmd/printf/printf.c
4818 printf(1) should support n$ width and precision specifiers
**********************************************************
```

```
/*
 * 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;                                                 \
        int dollar = 0;                                                 \
        if (*f == '$')  {                                               \
                dollar++;                                               \
                *f = '%';                                               \
        }                                                               \
```



```
        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);                                                \
        }                                                               \
        if (dollar)                                                     \
                *f = '$';                                               \
_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 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
```

```
121          * width or precision is a '*'; if it is, gather up value.  Note,
122          * format strings are reused as necessary to use up the provided
123          * arguments, arguments of zero/null string are provided to use
124          * up the format string.
125          */
126         fmt = format = *argv;
127         chopped = escape(fmt, 1, &len);         /* backslash interpretation */
128         rval = end = 0;
129         gargv = ++argv;
130
131         for (;;) {
132                 maxargv = gargv;
133                 char **maxargv = gargv;
134
135                 myargv = gargv;
136                 for (myargc = 0; gargv[myargc]; myargc++)
137                         /* nop */;
138                 start = fmt;
139                 while (fmt < format + len) {
140                         if (fmt[0] == '%') {
141                                 (void) fwrite(start, 1, PTRDIFF(fmt, start),
142                                     stdout);
143                                 if (fmt[1] == '%') {
144                                         /* %% prints a % */
145                                         (void) putchar('%');
146                                         fmt += 2;
147                                 } else {
148                                         fmt = doformat(fmt, &rval);
149                                         if (fmt == NULL)
150                                                 return (1);
151                                         end = 0;
152                                 }
153                                 start = fmt;
154                         } else
155                                 fmt++;
156                         if (gargv > maxargv)
157                                 maxargv = gargv;
158                 }
159                 gargv = maxargv;
160
161                 if (end == 1) {
162                         warnx1(_("missing format character"), NULL, NULL);
163                         return (1);
164                 }
165                 (void) fwrite(start, 1, PTRDIFF(fmt, start), stdout);
166                 if (chopped || !*gargv)
167                         return (rval);
168                 /* Restart at the beginning of the format string. */
169                 fmt = format;
170                 end = 1;
171         }
172         /* NOTREACHED */
173 }
174
175 static char *
176 doformat(char *fmt, int *rval)
177 doformat(char *start, int *rval)
178 {
179         static const char skip1[] = "#'-+ 0";
180         char **save;
181         static const char skip2[] = "0123456789";
182         char *fmt;
183         int fieldwidth, haveprec, havewidth, mod_ldbl, precision;
184         char convch, nextch;
185         char *start;
```



```
186         char *dptr;
187         int l;
188
189         start = alloca(strlen(fmt) + 1);
190         fmt = start + 1;
191
192         dptr = start;
193         *dptr++ = '%';
194         *dptr = 0;
195
196         fmt++;
197
198         /* look for "n$" field index specifier */
199         l = strspn(fmt, digits);
200         if ((l > 0) && (fmt[l] == '$')) {
201                 int idx = atoi(fmt);
202         fmt += strspn(fmt, skip2);
203         if ((*fmt == '$') && (fmt != (start + 1))) {
204                 int idx = atoi(start + 1);
205                 if (idx <= myargc) {
206                         gargv = &myargv[idx - 1];
207                 } else {
208                         gargv = &myargv[myargc];
209                 }
210                 if (gargv > maxargv) {
211                         maxargv = gargv;
212                 }
213                 fmt += l + 1;
214         }
215
216         save = gargv;
217
218         /* skip to field width */
219         while (strchr(skip1, *fmt) != NULL) {
220                 *dptr++ = *fmt++;
221                 *dptr = 0;
222         }
223
224         if (*fmt == '*') {
225
226                 start = fmt;
227                 fmt++;
228                 l = strspn(fmt, digits);
229                 if ((l > 0) && (fmt[l] == '$')) {
230                         int idx = atoi(fmt);
231                         if (idx <= myargc) {
232                                 gargv = &myargv[idx - 1];
233                         } else {
234                                 gargv = &myargv[myargc];
235         fmt = start + 1;
236                         }
237                         fmt += l + 1;
238                 }
239
240         /* skip to field width */
241         fmt += strspn(fmt, skip1);
242         if (*fmt == '*') {
243                 if (getint(&fieldwidth))
244                         return (NULL);
245                 if (gargv > maxargv) {
246                         maxargv = gargv;
247                 }
248                 havewidth = 1;
249
250                 *dptr++ = '*';
251                 *dptr = 0;
```

```
208                 ++fmt;
240         } else {
241                 havewidth = 0;

243                 /* skip to possible '.', get following precision */
244                 while (isdigit(*fmt)) {
245                         *dptr++ = *fmt++;
246                         *dptr = 0;
213                 fmt += strspn(fmt, skip2);
247                 }
248         }

250         if (*fmt == '.') {
251                 /* precision present? */
252                 fmt++;
253                 *dptr++ = '.';

217                 ++fmt;
255                 if (*fmt == '*') {

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

269                         if (getint(&precision))
270                                 return (NULL);
271                         if (gargv > maxargv) {
272                                 maxargv = gargv;
273                         }
274                         haveprec = 1;
275                         *dptr++ = '*';
276                         *dptr = 0;
222                         ++fmt;
277                 } else {
278                         haveprec = 0;

280                         /* skip to conversion char */
281                         while (isdigit(*fmt)) {
282                                 *dptr++ = *fmt++;
283                                 *dptr = 0;
227                         fmt += strspn(fmt, skip2);
284                         }
285                 }
286         } else
287                 haveprec = 0;
288         if (!*fmt) {
289                 warnx1(_("missing format character"), NULL, NULL);
290                 return (NULL);
291         }
292         *dptr++ = *fmt;
293         *dptr = 0;

295         /*
296          * Look for a length modifier.  POSIX doesn't have these, so
297          * we only support them for floating-point conversions, which
298          * are extensions.  This is useful because the L modifier can
299          * be used to gain extra range and precision, while omitting
300          * it is more likely to produce consistent results on different
```



```
301          * architectures.  This is not so important for integers
302          * because overflow is the only bad thing that can happen to
303          * them, but consider the command  printf %a 1.1
304          */
305         if (*fmt == 'L') {
306                 mod_ldbl = 1;
307                 fmt++;
308                 if (!strchr("aAeEfFgG", *fmt)) {
309                         warnx2(_("bad modifier L for %%%c"), *fmt, NULL);
310                         return (NULL);
311                 }
312         } else {
313                 mod_ldbl = 0;
314         }

316         gargv = save;
317         convch = *fmt;
318         nextch = *++fmt;

320         *fmt = '\0';
321         switch (convch) {
322         case 'b': {
323                 size_t len;
324                 char *p;
325                 int getout;

327                 p = strdup(getstr());
328                 if (p == NULL) {
329                         warnx2("%s", strerror(ENOMEM), NULL);
330                         return (NULL);
331                 }
332                 getout = escape(p, 0, &len);
333                 *(fmt - 1) = 's';
334                 PF(start, p);
335                 *(fmt - 1) = 'b';
336                 free(p);

338                 if (getout)
339                         return (fmt);
340                 break;
341         }
342         case 'c': {
343                 char p;

345                 p = getchr();
346                 PF(start, p);
347                 break;
348         }
349         case 's': {
350                 const char *p;

352                 p = getstr();
353                 PF(start, p);
354                 break;
355         }
356         case 'd': case 'i': case 'o': case 'u': case 'x': case 'X': {
357                 char *f;
358                 intmax_t val;
359                 uintmax_t uval;
360                 int signedconv;

362                 signedconv = (convch == 'd' || convch == 'i');
363                 if ((f = mknum(start, convch)) == NULL)
364                         return (NULL);
365                 if (getnum(&val, &uval, signedconv))
366                         *rval = 1;
```

```
367                 if (signedconv)
368                         PF(f, val);
369                 else
370                         PF(f, uval);
371                 break;
372         }
373         case 'e': case 'E':
374         case 'f': case 'F':
375         case 'g': case 'G':
376         case 'a': case 'A': {
377                 long double p;

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

unchanged_portion_omitted_