/\* memchr function \*/

#include <string.h>

void \*(memchr)(const void \*s, int c, size\_t n)

{ /\* find first occurrence of c in s[n] \*/

const unsigned char uc = c;

const unsigned char \*su = (const unsigned char \*)s;

for (; 0 < n; ++su, --n)

if (\*su == uc)

return ((void \*)su);

return (NULL);

}

/\* memcmp function \*/

#include <string.h>

int (memcmp)(const void \*s1, const void \*s2, size\_t n)

{ /\* compare unsigned char s1[n], s2[n] \*/

const unsigned char \*su1 = (const unsigned char \*)s1;

const unsigned char \*su2 = (const unsigned char \*)s2;

for (; 0 < n; ++su1, ++su2, --n)

if (\*su1 != \*su2)

return (\*su1 < \*su2 ? -1 : +1);

return (0);

}

/\* memcpy function \*/

#include <string.h>

void \*(memcpy)(void \*s1, const void \*s2, size\_t n)

{ /\* copy char s2[n] to s1[n] in any order \*/

char \*su1 = (char \*)s1;

const char \*su2 = (const char \*)s2;

for (; 0 < n; ++su1, ++su2, --n)

\*su1 = \*su2;

return (s1);

}

/\* memmove function \*/

#include <string.h>

void \*(memmove)(void \*s1, const void \*s2, size\_t n)

{ /\* copy char s2[n] to s1[n] safely \*/

char \*sc1 = (char \*)s1;

const char \*sc2 = (const char \*)s2;

if (sc2 < sc1 && sc1 < sc2 + n)

for (sc1 += n, sc2 += n; 0 < n; --n)

\*--sc1 = \*--sc2; /\*copy backwards \*/

else

for (; 0 < n; --n)

\*sc1++ = \*sc2++; /\* copy forwards \*/

return (s1);

}

/\* memset function \*/

#include <string.h>

void \*(memset)(void \*s, int c, size\_t n)

{ /\* store c throughout unsigned char s[n] \*/

const unsigned char uc = c;

unsigned char \*su = (unsigned char \*)s;

for (; 0 < n; ++su, --n)

\*su = uc;

return (s);

}

/\* strstr function \*/

#include <string.h>

char \*(strstr)(const char \*s1, const char \*s2)

{ /\* find first occurrence of s2[] in s1[] \*/

if (\*s2 == '\0')

return ((char \*)s1);

for (; (s1 = strchr(s1, \*s2)) != NULL; ++s1)

{ /\* match rest of prefix \*/

const char \*sc1, \*sc2;

for (sc1 = s1, sc2 = s2; ; )

if (\*++sc2 == '\0')

return ((char \*)s1);

else if (\*++sc1 != \*sc2)

break;

}

return (NULL);

}

/\* strcmp function \*/

#include <string.h>

int (strcmp)(const char \*s1, const char \*s2)

{ /\* compare unsigned char s1[], s2[] \*/

for (; \*s1 == \*s2; ++s1, ++s2)

if (\*s1 == '\0')

return (0);

return (\*(unsigned char \*)s1 < \*(unsigned char \*)s2

? -1 : +1);

}

/\* strlen function \*/

#include <string.h>

size\_t (strlen)(const char \*s)

{ /\* find length of s[] \*/

const char \*sc;

for (sc = s; \*sc != '\0'; ++sc)

;

return (sc - s);

}

/\* strcat function \*/

#include <string.h>

char \*(strcat)(char \*s1, const char \*s2)

{ /\* copy char s2[] to end of s1[] \*/

char \*s;

for (s = s1; \*s != '\0'; ++s)

; /\* find end of s1[] \*/

for (; (\*s = \*s2) != '\0'; ++s, ++s2)

; /\* copy s2[] to end \*/

return (s1);

}

/\* strchr function \*/

#include <string.h>

char \*(strchr)(const char \*s, int c)

{ /\* find first occurrence of c in char s[] \*/

const char ch = c;

for (; \*s != ch; ++s)

if (\*s == '\0')

return (NULL);

return ((char \*)s);

}

/\* strcoll function \*/

#include "xstrxfrm.h"

/\* type definitions \*/

typedef struct {

char buf[32];

const unsigned char \*s1, \*s2, \*sout;

\_Cosave state;

} Sctl;

static size\_t getxfrm(Sctl \*p)

{ /\* get transformed chars \*/

size\_t i;

do { /\* loop until chars delivered \*/

p->sout = (const unsigned char \*)p->buf;

i = \_Strxfrm(p->buf, &p->s1, sizeof (p->buf), &p->state);

if (0 < i && p->buf[i - 1] == '\0')

return (i - 1);

else if (\*p->s1 == '\0')

p->s1 = p->s2; /\* rescan \*/

} while (i == 0);

return (i);

}

int (strcoll)(const char \*s1, const char \*s2)

{ /\* compare s1[], s2[] using locale-dependent rule \*/

size\_t n1, n2;

Sctl st1, st2;

static const \_Cosave initial = {0};

st1.s1 = (const unsigned char \*)s1;

st1.s2 = (const unsigned char \*)s1;

st1.state = initial;

st2.s1 = (const unsigned char \*)s2;

st2.s2 = (const unsigned char \*)s2;

st2.state = initial;

for (n1 = n2 = 0; ; )

{ /\* compare transformed chars \*/

int ans;

size\_t n;

if (n1 == 0)

n1 = getxfrm(&st1);

if (n2 == 0)

n2 = getxfrm(&st2);

n = n1 < n2 ? n1 : n2;

if (n == 0)

return (n1 == n2 ? 0 : 0 < n2 ? -1 : +1);

else if ((ans = memcmp(st1.sout, st2.sout, n)) != 0)

return (ans);

st1.sout += n, n1 -= n;

st2.sout += n, n2 -= n;

}

}

/\* strcpy function \*/

#include <string.h>

char \*(strcpy)(char \*s1, const char \*s2)

{ /\* copy char s2[] to s1[] \*/

char \*s = s1;

for (s = s1; (\*s++ = \*s2++) != '\0'; )

;

return (s1);

}

/\* strcspn function \*/

#include <string.h>

size\_t (strcspn)(const char \*s1, const char \*s2)

{ /\* find index of first s1[i] that matches any s2[] \*/

const char \*sc1, \*sc2;

for (sc1 = s1; \*sc1 != '\0'; ++sc1)

for (sc2 = s2; \*sc2 != '\0'; ++sc2)

if (\*sc1 == \*sc2)

return (sc1 - s1);

return (sc1 - s1); /\* terminating nulls match \*/

}

/\* strerror function \*/

#include <errno.h>

#include <string.h>

char \*\_Strerror(int errcode, char \*buf)

{ /\* copy error message into buffer as needed \*/

static char sbuf[] = {"error #xxx"};

if (buf == NULL)

buf = sbuf;

switch (errcode)

{ /\* switch on known error codes \*/

case 0:

return ("no error");

case EDOM:

return ("domain error");

case ERANGE:

return ("range error");

case EFPOS:

return ("file positioning error");

default:

if (errcode < 0 || \_NERR <= errcode)

return ("unknown error");

else

{ /\* generate numeric error code \*/

strcpy(buf, "error #xxx");

buf[9] = errcode % 10 + '0';

buf[8] = (errcode /= 10) % 10 + '0';

buf[7] = (errcode / 10) % 10 + '0';

return (buf);

}

}

}

char \*(strerror)(int errcode)

{ /\* find error message corresponding to errcode \*/

return (\_Strerror(errcode, NULL));

}

/\* strncpy function \*/

#include <string.h>

char \*(strncpy)(char \*s1, const char \*s2, size\_t n)

{ /\* copy char s2[max n] to s1[n] \*/

char \*s;

for (s = s1; 0 < n && \*s2 != '\0'; --n)

\*s++ = \*s2++; /\* copy at most n chars from s2[] \*/

for (; 0 < n; --n)

\*s++ = '\0';

return (s1);

}

/\* strpbrk function \*/

#include <string.h>

char \*(strpbrk)(const char \*s1, const char \*s2)

{ /\* find index of first s1[i] that matches any s2[] \*/

const char \*sc1, \*sc2;

for (sc1 = s1; \*sc1 != '\0'; ++sc1)

for (sc2 = s2; \*sc2 != '\0'; ++sc2)

if (\*sc1 == \*sc2)

return ((char \*)sc1);

return (NULL); /\* terminating nulls match \*/

}

/\* strrchr function \*/

#include <string.h>

char \*(strrchr)(const char \*s, int c)

{ /\* find last occurrence of c in char s[] \*/

const char ch = c;

const char \*sc;

for (sc = NULL; ; ++s)

{ /\* check another char \*/

if (\*s == ch)

sc = s;

if (\*s == '\0')

return ((char \*)sc);

}

}

/\* strspn function \*/

#include <string.h>

size\_t (strspn)(const char \*s1, const char \*s2)

{ /\* find index of first s1[i] that matches no s2[] \*/

const char \*sc1, \*sc2;

for (sc1 = s1; \*sc1 != '\0'; ++sc1)

for (sc2 = s2; ; ++sc2)

if (\*sc2 == '\0')

return (sc1 - s1);

else if (\*sc1 == \*sc2)

break;

return (sc1 - s1); /\* null doesn't match \*/

}

/\* strtok function \*/

#include <string.h>

char \*(strtok)(char \*s1, const char \*s2)

{ /\* find next token in s1[] delimited by s2[] \*/

char \*sbegin, \*send;

static char \*ssave = ""; /\* for safety \*/

sbegin = s1 ? s1 : ssave;

sbegin += strspn(sbegin, s2);

if (\*sbegin == '\0')

{ /\* end of scan \*/

ssave = ""; /\* for safety \*/

return (NULL);

}

send = sbegin + strcspn(sbegin, s2);

if (\*send != '\0')

\*send++ = '\0';

ssave = send;

return (sbegin);

}

/\* strxfrm function \*/

#include "xstrxfrm.h"

size\_t (strxfrm)(char \*s1, const char \*s2, size\_t n)

{ /\* transform s2[] to s1[] by locale-dependent rule \*/

size\_t nx = 0;

const unsigned char \*s = (const unsigned char \*)s2;

\_Cosave state = {0};

while (nx < n)

{ /\* translate and deliver \*/

size\_t i = \_Strxfrm(s1, &s, n - nx, &state);

s1 += i, nx += i;

if (0 < i && s1[-1] == '\0')

return (nx - 1);

else if (\*s == '\0')

s = (const unsigned char \*)s2; /\* rescan \*/

}

for (; ; )

{ /\* translate and count \*/

char buf[32];

size\_t i = \_Strxfrm(buf, &s, sizeof (buf), &state);

nx += i;

if (0 < i && buf[i - 1] == '\0')

return (nx - 1);

else if (\*s == '\0')

s = (const unsigned char \*)s2; /\* rescan \*/

}

}

/\* \_Strxfrm function \*/

#include <limits.h>

#include "xstrxfrm.h"

size\_t \_Strxfrm(char \*sout, const unsigned char \*\*psin,

size\_t size, \_Cosave \*ps)

{ /\* translate string to collatable form \*/

unsigned char state = ps->\_State;

int leave = 0;

int limit = 0;

int nout = 0;

const unsigned char \*sin = \*psin;

unsigned short wc = ps->\_Wchar;

for (; ; )

{ /\* perform a state transformation \*/

unsigned short code;

const unsigned short \*stab;

if (\_NSTATE <= state

|| (stab = \_Costate.\_Tab[state]) == NULL

|| (\_NSTATE\*UCHAR\_MAX) <= ++limit

|| (code = stab[\*sin]) == 0)

break;

state = (code & ST\_STATE) >> ST\_STOFF;

if (code & ST\_FOLD)

wc = wc & ~UCHAR\_MAX | code & ST\_CH;

if (code & ST\_ROTATE)

wc = wc >> CHAR\_BIT & UCHAR\_MAX | wc << CHAR\_BIT;

if (code & ST\_OUTPUT && ((sout[nout++]

= code & ST\_CH ? code : wc) == '\0'

|| size <= nout))

leave = 1;

if (code & ST\_INPUT)

if (\*sin != '\0')

++sin, limit = 0;

else

leave = 1;

if (leave)

{ /\* return for now \*/

\*psin = sin;

ps->\_State = state;

ps->\_Wchar = wc;

return (nout);

}

}

sout[nout++] = '\0'; /\* error return \*/

\*psin = sin;

ps->\_State = \_NSTATE;

return (nout);

}

/\* strncat function \*/

#include <string.h>

char \*(strncat)(char \*s1, const char \*s2, size\_t n)

{ /\* copy char s2[max n] to end of s1[] \*/

char \*s;

for (s = s1; \*s != '\0'; ++s)

; /\* find end of s1[] \*/

for (; 0 < n && \*s2 != '\0'; --n)

\*s++ = \*s2++; /\* copy at most n chars from s2[] \*/

\*s = '\0';

return (s1);

}

/\* strncmp function \*/

#include <string.h>

int (strncmp)(const char \*s1, const char \*s2, size\_t n)

{ /\* compare unsigned char s1[max n], s2[max n] \*/

for (; 0 < n; ++s1, ++s2, --n)

if (\*s1 != \*s2)

return (\*(unsigned char \*)s1

< \*(unsigned char \*)s2 ? -1 : +1);

else if (\*s1 == '\0')

return (0);

return (0);

}

/\* atoi function \*/

#include <stdlib.h>

int (atoi)(const char \*s)

{ /\* convert string to int \*/

return ((int)\_Stoul(s, NULL, 10));

}

/\* strtoul function \*/

#include <stdlib.h>

/\* \_Stoul function \*/

#include <stdlib.h>

#include <ctype.h>

#include <errno.h>

#include <limits.h>

#include <stddef.h>

#include <string.h>

/\* macros \*/

#define BASE\_MAX 36 /\* largest valid base \*/

/\* static data \*/

static const char digits[] = { /\* valid digits \*/

"0123456789abcdefghijklmnopqrstuvwxyz"};

static const char ndigs[BASE\_MAX+1] = { /\* 32-bits! \*/

0, 0, 33, 21, 17, 14, 13, 12, 11, 11,

10, 10, 9, 9, 9, 9, 9, 8, 8, 8,

8, 8, 8, 8, 7, 7, 7, 7, 7, 7,

7, 7, 7, 7, 7, 7, 7,};

unsigned long \_Stoul(const char \*s, char \*\*endptr, int base)

{ /\* convert string to unsigned long, with checking \*/

const char \*sc, \*sd;

const char \*s1, \*s2;

char dig, sign;

ptrdiff\_t n;

unsigned long x, y;

for (sc = s; isspace(\*sc); ++sc)

;

sign = \*sc == '-' || \*sc == '+' ? \*sc++ : '+';

if (base < 0 || base == 1 || BASE\_MAX < base)

{ /\* silly base \*/

if (endptr)

\*endptr = (char \*)s;

return (0);

}

else if (0 < base)

{ /\* strip 0x or 0X \*/

if (base == 16 && \*sc == '0'

&& (sc[1] == 'x' || sc[1] == 'X'))

sc += 2;

}

else if (\*sc != '0')

base = 10;

else if (sc[1] == 'x' || sc[1] == 'X')

base = 16, sc += 2;

else

base = 8;

for (s1 = sc; \*sc == '0'; ++sc)

; /\* skip leading zeros \*/

x = 0;

for (s2 = sc; (sd = (char \*)memchr(digits,

tolower(\*sc), base)) != NULL; ++sc)

{ /\* accumulate digits \*/

y = x, dig = sd - digits; /\* for overflow checking \*/

x = x \* base + dig;

}

if (s1 == sc)

{ /\* check string validity \*/

if (endptr)

\*endptr = (char \*)s;

return (0);

}

n = sc - s2 - ndigs[base];

if (n < 0)

;

else if (0 < n || x < x - dig

|| (x - dig) / base != y)

{ /\* overflow \*/

errno = ERANGE;

sc = s, x = ULONG\_MAX, sign = '+';

}

if (sign == '-') /\* get final value \*/

x = -x;

if (endptr)

\*endptr = (char \*)sc;

return (x);

}