stcpm, stcpma

Pattern match functions

Class: Lattice *Category: String Search*

SYNOPSIS

```
#include  <string.h>

size  =  stcpm(string,pattern,match);
                                Unanchored  pattern  match
size  =  stcpma(string,pattern); Anchored  patter  match

size_t  size;              size  of  matching  string
const  char  *string;      string  to  be  scanned
const  char  *pattern;     pattern  string
char  **match;             returns  pointer  to  matching
                           string
```

DESCRIPTION

These functions scan a string to find a specified pattern. The pattern is specified in a simplified form of regular expression notation as shown below:

Pattern	Meaning
?	Match any single character
c*	Match 0 or more instances of character c
c+	Match 1 or more instances of character c
\?	Match a question mark (?)
*	Match an asterisk (*)
\+	Match a plus sign (+)

Any other character must match exactly. For example,

Pattern	Matching
"abc"	Only "abc"
"ab*c"	"ac" or "abc" or "abbc" and so on
"ab+c"	"abc" or "abbc" or "abbbc" and so on
"ab?*c"	Any string starting with "ab" and ending with "c"
"ab*c"	Only "ab*c"

Notice that the last pattern requires a double backslash in front of the asterisk. This causes the compiler to place a single backslash in the string so that stcpm or stcpma will see the string as "ab*c".

For stcpma, the match must occur at the beginning of the string, while for stcpm, the match can occur anywhere in the string. In either case, the function returns the size of the matching string or zero if there was no match. Also, stcpm returns a pointer to the beginning of the matching string.

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>

int  main(void)
{
  char  s[100],p[100],*r;
  int  x;

  for  (;;)
  {
    printf("\nSearch  string  =>  ");
    if(gets(s)  ==  NULL)
      break;
    printf("Pattern  =>  ");
    if(gets(p)  ==  NULL)
      break;
    x  =  stcpma(s,p);
    if(x)
      printf("stcpma:  %d,  \"%.*s\"\n",x,x,s);
    else
      printf("stcpma:  no  match\n");
    x  =  stcpm(s,p,&r);
    if(x)
      printf("stcpm:  %d,  \"%.*s\"\n",x,x,r);
    else
      printf("stcpm:  no  match\n");
  }
  return  0
}
```

stpblk — Skip blanks (white space)

Class: Lattice *Category: String Search*

SYNOPSIS

```
#include  <string.h>

q  =  stpblk(p);

char  *q;             updated  string  pointer
const  char  *p;      string  pointer
```

DESCRIPTION

This function advances the string pointer past white space characters, that is, past all the characters for which `isspace` is true.

RETURNS

The function returns a pointer to the next non-white-space character. Note that the null terminator byte is not considered to be white space, and so the function will not go past the end of the string.

SEE

stcis, strspn

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>

int  main(void)
{
  char  input[256];

  for  (;;)
  {
    puts("\nEnter  a  string  with  leading  blanks...");
    if(gets(input)  ==  NULL)
      break;
    printf("%s\n",stpblk(input));
  }
  return  0
}
```

stpdate — Convert date array to string

Class: Lattice *Category: Date and Time*

SYNOPSIS

```
#include  <string.h>

np  =  stpdate(p,mode,date);

char  *np;              updated  output  string  pointer
char  *p;               output  string  pointer
int  mode;              conversion  mode
const  char  *date;     date  array,  as  follows
                        date[0]  =>  year  -  1980
                        date[1]  =>  month  (1  to  12)
                        date[2]  =>  day  (1  to  31)
```

DESCRIPTION

This function converts a 3-byte date array into ASCII or BCD according to the `mode` argument:

Mode	Date Format
0	yymmdd (BCD, 3 bytes)
1	yymmdd (ASCII, 7 bytes)
2	mm/dd/yy (ASCII, 9 bytes)
3	mm-dd-yy (ASCII, 9 bytes)
4	MMM d, yyyy (ASCII, up to 13 bytes)
5	Mm...m d, yyyy (ASCII, up to 19 bytes)
6	dd MMM yy (ASCII, 10 bytes)
7	dd MMM yyyy (ASCII, 12 bytes)

In the above formats, `MMM` represents a 3-character month abbreviation in capitals, and `Mm...m` represents the full month name (e.g. January). The `mm`, `dd`, and `yy` terms are 2-character month, day, and year, respectively, while `d` is the date with the leading zero suppressed. The `yyyy` term is the 4-character year obtained by adding 1980 to the first byte of the date array.

For all modes except 0, a null byte is appended to the output string.

RETURNS

The function does not make validity checks on the date array, and so it cannot fail. It returns a pointer to the first byte past the generated output. For modes other than 0, this is a pointer to the null terminator.

SEE

stptime, getclk, getft, ftunpk

stpsym

Get next symbol from a string

Class: Lattice *Category: String Search*

SYNOPSIS

```
#include <string.h>

p = stpsym(s,sym,symlen);

char *p;            points to next input character
const char *s;      input string
char *sym;          output string
size_t symlen;      sizeof(sym)
```

DESCRIPTION

This function extracts the next symbol from the input string. The first character of the symbol must be alphabetic (upper or lower case), and the remaining characters must be alphanumeric. Note that the pointer is not advanced past any initial white space in the input string.

The output string is the null-terminated symbol, and will be an empty string if no symbol is found. If the symbol is longer than symlen-1, its excess characters are dropped.

RETURNS

The function returns a pointer to the next character past the symbol.

SEE

stcarg, stpbrk, strcspn, strpbrk

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>

int  main(void)
{
  char  a[256],b[10];
  char  *p;

  for  (;;)
  {
    printf("\nEnter  text  string:  ");
    if(gets(a)  ==  NULL)
      break;
    for  (;;)
    {
      p  =  stpsym(a,b,sizeof(b));
      printf("Symbol:  \"%s\"  Residual:  \"%s\"\n",b,p);
      if(b[0]  ==  '\0')
        break;
    }
  }
  return  0;
}
```

stptime

Convert time array to string

Class: Lattice *Category: Date and Time*

SYNOPSIS

```
#include  <string.h>

np  =  stptime(p,mode,time);

char  *np;               updated  output  string  pointer
char  *p;                output  string  pointer
int  mode;               conversion  mode
const  char  *time;      time  array,  as  follows
                         time[0]  =>  hour  (0  to  23)
                         time[1]  =>  minute  (0  to  59)
                         time[2]  =>  second  (0  to  59)
                         time[3]  =>  hundredths  (0  to  99)
```

DESCRIPTION

This function converts a 4-byte time array into ASCII or BCD according to the mode argument:

Mode	Time Format
0	hhmmssdd (BCD, 4 bytes)
1	hhmmss (ASCII, 7 bytes)
2	hh:mm:ss (ASCII, 9 bytes)
3	hhmmssdd (ASCII, 9 bytes)
4	hh:mm:ss.dd (ASCII, 12 bytes)
5	hh:mm (ASCII, 6 bytes)
6	hr:mm:ss HH (ASCII, 12 bytes)
7	hr:mm HH (ASCII, 9 bytes)

The hh, mm, ss, and dd terms are simply the 2-digit (BCD or ASCII) equivalents of the binary values in the time array. The hr term is the 2-digit hour using the 12-hour form, and the HH term is either AM or PM.

Note that a null terminator is appended to the ASCII output strings.

RETURNS

The function does not make validity checks on the time array, and so it cannot fail. It returns a pointer to the first byte past the generated output. For modes other than 0, this is a pointer to the null terminator.

SEE

stpdate, getclk, getft, ftunpk

stptok — Get next token from a string

Class: Lattice *Category: String Search*

SYNOPSIS

```
#include <string.h>

p = stptok(s,tok,toklen,brk);

char *p;                points to next character after
                        token
const char *s;          points to input string
char *tok;              points to output buffer
size_t toklen;          sizeof(tok)
const char *brk;        break string
```

DESCRIPTION

This function breaks out the next token from the input string and moves it to the token buffer with a null terminator. A token consists of all characters in the input string s up to but not including the first character that is in the break string. In other words, brk specifies the characters that cannot be included in a token.

If the input string begins with a break character, then the token buffer will contain a null string, and the return pointer p will be the same as s. If no break character is found after toklen-1 input characters have been moved to the token buffer, or if the input string terminator (a null byte) is hit, then the scan stops as if a break character were hit.

RETURNS

The function returns a pointer to the next character in the input string.

Note that the function does not delete white space at the beginning of the input string.

SEE

stpblk, strtok

strbpl — Build string pointer list

Class: Lattice *Category: String Search*

SYNOPSIS

```
#include  <string.h>

n  =  strbpl(s,max,t);

long  n;               number  of  pointers
char  *s[];            pointer  to  string  pointer  list
size_t  max;           maximum  number  of  pointers
const  char  *t;       text  pointer
```

DESCRIPTION

This function constructs a list of pointers to the strings contained within the specified text array. Each string must be null-terminated, and the text array must be terminated by a null string. In other words, array t must end with two null bytes, one to terminate the final string and another to terminate the array. The string pointer list s is terminated by a null pointer.

RETURNS

The return value indicates how many string pointers were placed into the array s, not including the NULL terminator If the number of strings plus the final null pointer is greater than max, a value of -1 is returned.

SEE

getfnl, strsrt

strcat, strncat — Concatenate strings

Class: ANSI *Category: String Copy*

SYNOPSIS

```
#include  <string.h>

p  =  strcat(to,from);
p  =  strncat(to,from,n);

char  *p;                 same  as  destination  string  pointer
char  *to;                destination  string  pointer
const  char  *from;       source  string  pointer
size_t  n;                length  count
```

DESCRIPTION

The strcat function concatenates the source string to the tail end of the destination string. Compare this function with strncat, which allows you to specify the maximum number of characters which will be added.

A null byte is placed at the end of the destination.

RETURNS

The strcat and strncat functions return a pointer that is the same as the first argument.

SEE

strcpy, stpcpy, strncpy

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>

int  main(void)
{
  char  a[256],b[256];
  long  n;

  for  (;;)
  {
    printf("\nEnter  string  A:  ");
    if(gets(a)  ==  NULL)
      break;
    printf("Enter  string  B:  ");
    if(gets(b)  ==  NULL)
      break;
    printf("Enter  maximum  length  N:  ");
    scanf("%ld",&n);
    printf("strcat(A,B):  \"%s\"\n",strcat(a,b));
    printf("strncat(A,B,N):  \"%s\"\n",strncat(a,b,n));
  }
  return  0;
}
```

strchr, strrchr, stpchr, stpchrn Find character

Class: ANSI *Category: String Search*

SYNOPSIS

```
#include  <string.h>

p  =  stpchr(s,c);      find  character  in  string
p  =  stpchrn(s,c);     find  character  not  in  string
p  =  strchr(s,c);      find  character  in  string
p  =  strrchr(s,c);     find  last  character  in  string

char  *p;               updated  string  pointer
const  char  *s;        input  string  pointer
int  c;                 character  to  be  located
```

DESCRIPTION

The stpchr and strchr functions scan the input string to find the first occurrence of the character specified by argument c. Similarly, stpchrn scans for the first occurrence of some character other than c. The strrchr function scans the input string to find the last occurrence of the character specified by argument c.

stpchr is provided for compatibility with other versions of Lattice C, whilst the strchr function is now part of the ANSI standard.

RETURNS

For strchr, strrchr and stpchr a NULL pointer is returned if the input string is empty or if the specified character is not found. stpchrn returns a NULL pointer if the input string is empty or consists entirely of character c.

strcmp, stricmp, strncmp, strnicmp

Compare strings

Class: ANSI *Category: String Comparison*

SYNOPSIS

```
#include  <string.h>

x  =  strcmp(a,b);      Compare  strings
x  =  stricmp(a,b);     Compare  strings,  case-
                        insensitive
x  =  strncmp(a,b,n);   Compare  strings,  length-limited
x  =  strnicmp(a,b,n);  Compare  strings,  no  case,  max
                        size

int  x;                 comparison  result
const  char  *a,*b;     strings  being  compared
size_t  n;              length  limiter
```

DESCRIPTION

These functions compare two null-terminated strings. The ASCII collating sequence is used in all cases, but stricmp and strnicmp do not distinguish between upper and lower case. Note also that stricmp and strnicmp are not part of the ANSI C standard.

The relative collating sequence of the strings is indicated by the sign of the return value, as follows:

Return	Meaning
Negative	First string is below second
Zero	Strings are equal
Positive	First string is above second

If the strings have different lengths, the shorter one is treated as if it were extended with zeroes. For strncmp and strnicmp, no more than n characters are compared.

Note that strcmp has a built-in version which is functionally equivalent to the standard library version. The statement #include <string.h> provides a default setting by which built-in functions are accessed. If you don't want the built-in function, you can use an #undef statement i.e. #undef strcmp.

RETURNS

As noted above.

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>

void  result(const  char  *name,  size_t  r)
{
  char  *p;

  if(r  ==  0)
    p  =  "is  equal  to";
  else  if(r  <  0)
    p  =  "is  less  than";
  else  if(r  >  0)
    p  =  "is  greater  than";
  printf("%s  String  A  %s  string  B\n",name,p);
}

int  main(void)
{
  char  a[256],b[256];
  long  n;

  for  (;;)
  {
    printf("Enter  string  A:  ");
    if(gets(a)  ==  NULL)
      break;
    printf("Enter  string  B:  ");
    if(gets(b)  ==  NULL)
      break;
    printf("Enter  maximum  compare  length:  ");
    scanf("%d",&n);

    result("strcmp:   ",strcmp(a,b));
    result("stricmp:  ",stricmp(a,b));
    result("strncmp:  ",strncmp(a,b,n));
    result("strnicmp:",strnicmp(a,b,n));
  }
  return  0;
}
```

strcoll — Locale-specific string comparison

Class: ANSI — *Category: String Comparison*

SYNOPSIS

```
#include <string.h>

num = strcoll(s1,s2);

int num;            integer indicating comparison result
const char *s1;     first string to be compared
const char *s2;     second string to be compared
```

DESCRIPTION

The strcoll function compares the string pointed to by s1 with the string pointed to by s2, with both interpreted as appropriate to the LC_COLLATE (defined in locale.h) category of the current locale.

The strcoll and strxfrm functions provide locale-specific string sorting. The former is intended for applications in which the number of comparisons is small, while the latter is more appropriate when items are to be compared a number of times; the cost of transformation is then only paid once.

RETURNS

The strcoll function returns an integer greater than, equal to, or less than zero, as the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 when both are interpreted as appropriate to the current locale.

SEE

strxfrm

strcpy, strncpy, stccpy, stpcpy — Copy strings

Class: ANSI — *Category: String Copy*

SYNOPSIS

```
#include <string.h>

p = strcpy(to,from);
p = strncpy(to,from,n);
size = stccpy(to,from,n);
np = stpcpy(to,from);

char *np;               points to end of destination
                        string
char *p;                same as destination pointer
char *to;               destination pointer
const char *from;       source pointer
size_t n;               maximum source length
size_t size;            number of bytes copied
```

DESCRIPTION

These functions copy the null-terminated source string to the destination area. For stpcpy and strcpy, the entire source string is copied, and the resulting destination is always null-terminated. The strncpy function always writes *exactly* n characters to the destination. If the null terminator is hit before n characters are copied from the source, then the destination is filled with null bytes. If the source string contains more than n non-null characters, the destination will not be null-terminated.

The stccpy function is similar to strncpy except that it always produces a null-terminated string, and it returns the actual number of bytes (size) placed in the to area, including the null terminator. Note that it may copy less than n bytes.

Note that strcpy has a built-in version which is functionally equivalent to the standard library version. The statement #include <string.h> provides a default setting by which built-in functions are accessed. If you don't want the built-in function, you can use an #undef statement i.e. #undef strcpy.

Note that stpcpy and stccpy do not form part of the ANSI C standard, also note that you should be careful when using strncpy, since it is one of the few string functions which does not produce a null-terminated string under every condition.

RETURNS

The strcpy and strncpy functions return a pointer that is the same as the destination pointer. The Lattice function stpcpy returns a pointer to the end of the destination string, which is often more useful when you are building a string up from several pieces.

EXAMPLE

```
/*
 * This example should print: Hello, my name is John.
 *
 */
#include <string.h>

int main(void)
{
  char b[256],*p;

  p = stpcpy(b,"Hello, ");
  p = stpcpy(p,"my name is ");
  p = stpcpy(p,"John.");
  puts(b);
  return 0;
}
```

strdup

Duplicate a string

Class: XENIX *Category: String Copy*

SYNOPSIS

```
#include <string.h>

p = strdup(s);

char *p;            points to duplicate string
const char *s;      points to string being duplicated
```

DESCRIPTION

This function creates a duplicate of the specified string by using malloc and strcpy to allocate space and copy the string to it.

RETURNS

A NULL pointer is returned if malloc fails. Otherwise, the function returns a pointer to the duplicate string.

strerror — Map error number in errnum to error message

Class: ANSI *Category: Errors*

SYNOPSIS

```
    #include  <string.h>

    errmsg  =  strerror(errnum);

    char  *errmsg;     error  message  string
    int  errnum;       error  number
```

DESCRIPTION

The strerror function maps the value in errnum to an error message string pointed to by errmsg.

RETURNS

The strerror function returns a pointer to the string, the contents of which is an error message. The array pointed to cannot be modified by the program, but may be overwritten by a subsequent call to the strerror function.

strftime — Format using locale control parameters

Class: ANSI *Category: Date and Time*

SYNOPSIS

```
    #include  <time.h>

    ret  =  strftime(s,maxsize,format,timeptr);

    size_t  ret;            0  if  successful
    char  *s;               array  to  contain  characters
    size_t  maxsize;        maximum  number  of  characters
    const  char  *format;  specifier  to  control  formatting
    const  struct  tm  *timeptr;
                            time  values
```

DESCRIPTION

The strftime function places characters into the array pointed to by s as controlled by the string pointed to by format. The format is a multibyte character sequence, beginning and ending in its initial shift state. The format string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a % character followed by a character that determines the behaviour of the conversion specifier. All ordinary multibyte characters (including the terminating null character) are copied unchanged into the array. No more than maxsize characters are placed into the array. Each conversion specifier is replaced by appropriate characters as described later. The appropriate characters are determined by the LC_TIME category of the current locale and by the values contained in the structure pointed to by timeptr.

The strftime function provides a way of formatting the date and time in the appropriate locale-specific fashion, using the %c, %x, and %X format specifiers. More generally, it allows the programmer to tailor whatever date and time format is appropriate for a given application. The facility is based on the UNIX system date command, by which each conversion specifier is replaced by appropriate characters described in the following list:

Code	Replaced by
%a	the locale's abbreviated weekday name
%A	the locale's full weekday name
%b	the locale's abbreviated month name
%B	the locale's full month name
%c	the locale's appropriate date and time representation

%d	the day of the month as a decimal number (01-31)
%H	the hour (24-hour clock) as a decimal number (00-23)
%I	the hour (12-hour clock) as a decimal number (01-12)
%j	the day of the year as a decimal number (001-366)
%m	the day of the month as a decimal number (01-31)
%M	the minute as a decimal number (00-59)
%p	the locale's equivalent of the AM/PM designations associated with a 12-hour clock
%S	the second as a decimal number (00-61)
%U	the week number of the year (the first Sunday as the first day of week 1) as a decimal number (00-53)
%w	the weekday as a decimal number with Sunday as 0 (0-6)
%W	the week number of the year (the first Monday as the first day of week 1) as a decimal number (00-53)
%x	the locale's appropriate date representation
%X	the locale's appropriate time representation
%y	the year without century as a decimal number (00-99)
%Y	the year with century as a decimal number
%Z	the time zone name or abbreviation, or by no characters if no time zone is determinable
%%	two % characters are required to specify a single %

RETURNS

If the total number of resulting characters including the terminating null character is not more than maxsize, the strftime function returns the number of characters placed into the array pointed to by s not including the terminating null character. Otherwise, zero is returned and the contents of the array are truncated to maxsize characters and will not be null ('\0') terminated.

strins

Insert a string

Class: Lattice *Category: String Copy*

SYNOPSIS

```
#include <string.h>

strins(to,from);

char *to;                destination string
const char *from;        source string
```

DESCRIPTION

This function inserts the source string (to) in front of the destination string (from). Both strings must be null-terminated, and the destination is shifted to the right (upward in memory) in order to accomodate the source string. The final result is a single null-terminated string.

SEE

strcat

EXAMPLE

```
#include <string.h>

char here[] = "Here ";
char now[30] = "and now";

strins(now,here);      /* now => "Here and now" */
```

strlen, stclen

Measure length of a string

Class: ANSI

Category: String Copy

SYNOPSIS

```
    #include  <string.h>

    length  =  strlen(s);    Measure  length  of  a  string
    length  =  stclen(s);    Measure  length  of  a  string

    const  char  *s;
    size_t  length;      number  of  bytes  in  s  (before  NULL)
```

DESCRIPTION

These functions return the number of bytes in string s before the null terminator byte. The strlen function is the ANSI equivalent of the Lattice implementation stclen.

Note that strlen has a built-in version which is functionally equivalent to the standard library version. The statement #include <string.h> provides a default setting by which built-in functions are accessed. If you don't want the built-in function, you can use an #undef statement as i.e. #undef strlen.

RETURNS

The number of bytes in the string s before the null byte.

strlwr, strupr

Change case of string

Class: XENIX

Category: String Conversion

SYNOPSIS

```
    #include  <string.h>

    p  =  strlwr(s);          convert  string  to  lower  case
    p  =  strupr(s);          convert  string  to  upper  case

    char  *p;                 return  pointer  (same  as  s)
    char  *s;                 string  pointer
```

DESCRIPTION

These functions convert all alphabetic characters in the specified null-terminated string to lower or upper case. In each case, the function return value is the same as the string pointer.

RETURNS

Both functions return the original string pointer.

strmfe

Make file name with extension

Class: Lattice

Category: File Name Manipulation

SYNOPSIS

```
#include  <string.h>

strmfe(newname,oldname,ext);

char  *newname;               new  file  name
const  char  *oldname;        old  file  name
const  char  *ext;            extension
```

DESCRIPTION

This function copies the old file name to the new name, deleting any extension. Then it appends the specified extension to the new file name, with an intervening period. For example,

Oldname	Ext	Newname
"c:myprog.c"	"cc"	"c:myprog.cc"
"abc"	"prg"	"abc.prg"

The newname area must be large enough to accept the file name string and the separator. A safe size is FMSIZE, which is defined in the dos.h header file.

SEE

strmfn, strmfp

strmfn

Make file name from components

Class: Lattice

Category: File Name Manipulation

SYNOPSIS

```
#include  <string.h>

strmfn(file,drive,path,node,ext);

char  *file;                file  name  pointer
const  char  *drive;        drive  code  pointer
const  char  *path;         directory  path  pointer
const  char  *node;         node  pointer
const  char  *ext;          extension  pointer
```

DESCRIPTION

This function makes a file name from four possible components. In general, the name is constructed as follows:

```
drive:path\node.ext
```

If the drive pointer is not NULL, that string is moved to the area pointed to by the file argument. Then a colon is inserted unless one is already there. Next, if path is not NULL, it is appended to file, and the directory separator specified by _SLASH is added if necessary. The node string is appended next, unless it is NULL. Finally, if ext is not NULL, a period is appended to file, followed by the ext string.

RETURNS

None. Make sure that the file pointer refers to an area that is large enough to hold the result. A safe value is FMSIZE, which is defined in dos.h.

SEE

strmfe, strmfp, _SLASH

strmfp

Make file name from path/node

Class: Lattice *Category: File Name Manipulation*

SYNOPSIS

```
    #include  <string.h>

    strmfp(name,path,node);

    char  *name;            file  name
    const  char  *path;     directory  path
    const  char  *node;     node
```

DESCRIPTION

This function copies the path string to the file name area, appending the _SLASH separator if the path string is not empty and does not end with a slash, backslash, or colon. Then the node string is appended to the file name. _SLASH is an external character variable that defaults to a backslash (\).

The name area must be large enough to accept the file name string. A safe value is FMSIZE, which is defined in the dos.h header file.

SEE

strmfe, strmfn, _SLASH

strmid

Return a substring from a string

Class: Lattice *Category: String Copy*

SYNOPSIS

```
    #include  <string.h>

    error  =  strmid(source,dest,pos,len);

    char  *dest;              destination  pointer
    const  char  *source;     source  pointer
    size_t  pos;              starting  position  of  dest  in
                              source
    size_t  len;              length  of  substring
    int  error;               -1  if  pos  is  beyond  source,
                              else  0
```

DESCRIPTION

The strmid function returns a pointer to a substring of source beginning at character position pos, and having a length of len. If len is greater than the length of source offset at pos, then the rest of the string is copied to dest.

The destination string is null-terminated.

RETURNS

If pos is beyond the length of source, then -1 is returned. Otherwise, 0 is returned.

SEE

strins

strpbrk,stpbrk

Find break character in string

Class: ANSI *Category: String Search*

SYNOPSIS

```
#include  <string.h>

p  =  stpbrk(s,b);
p  =  strpbrk(s,b);

char  *p;           points  to  break  character  in  s
const  char  *s;    string  to  be  scanned
const  char  *b;    break  characters
```

DESCRIPTION

These functions scan string s to find the first occurrence of a character from break string b. They are completely equivalent, except that strpbrk is the ANSI name, while stpbrk is the traditional Lattice name.

RETURNS

If no character from b is found in s, a NULL pointer is returned. Otherwise, p is a pointer to the break first break character.

SEE

strspn, strcspn

EXAMPLE

```
#include  <string.h>
#include  <stdio.h>

/*
 *  Scan  for  commas,  periods,  and  blanks.  Display  the
 *  tail  of  the  string  each  time  a  break  character  is
 *  found.
*/
char  *p,s[  ]  =  "Hello,  I  must  be  going.";

for(p  =  s;  p  =  strbrk(p,",.  ");  )
  printf("%s\n",p);
```

strrev

Reverse a character string

Class: XENIX *Category: String Copy*

SYNOPSIS

```
#include  <string.h>

p  =  strrev(s);

char  *p,*s;  string  pointer
```

DESCRIPTION

This function reverses a character string. That is, it "reflects" the string about its mid-point such that the last character is first and the first is last.

RETURNS

This function returns the same pointer that was passed to it.

EXAMPLE

```
char  *s="Rotavator";

printf("%s  reversed  is  ",s);
strrev(s);
printf("%s\n",s);

/*  will  print  "Rotavator  reversed  is  rotavatoR"  */
```

strset, strnset — Set string to value

Class: XENIX *Category: String Copy*

SYNOPSIS

```
#include <string.h>

p = strset(s,c);
p = strnset(s,c,n);

char *p;        return pointer (same as s)
char *s;        string pointer
int c;          value
size_t n;       maximum string length
```

DESCRIPTION

The strset and strnset functions set all bytes of a null-terminated string to the same value, not including the terminator byte. With the strnset function, you can specify a maximum length in bytes, given by n.

RETURNS

The original string pointer is returned.

strsfn — Split file name

Class: Lattice *Category: File Name Manipulation*

SYNOPSIS

```
#include <string.h>

strsfn(file,drive,path,node,ext);

const char *file;   file name pointer
char *drive;        drive code pointer
char *path;         directory path pointer
char *node;         node pointer
char *ext;          extension pointer
```

DESCRIPTION

This function splits a file name into four possible components and places them into the drive, path, node, and ext strings. If any of those arguments are NULL, then those components are discarded.

In general, a complete file name is constructed as follows:

```
drive:path\node.ext
```

When strsfn splits the file name, it leaves the colon attached to the drive code, but removes trailing punctuation from the other components. Slashes or backslashes within the path component are preserved. If the file name is of the form

```
drive:\node.ext
```

then the path component is a single backslash.

RETURNS

You must make sure that the drive, path, node, and ext pointer refer to areas that are large enough to hold the largest string that might be generated. The following lengths are safe:

Part	Size
drive	3 bytes
path	FMSIZE in dos.h
node	FNSIZE in dos.h
ext	FESIZE in dos.h

This function does not check that these lengths are not exceeded, although it does copy file string to an internal buffer of size FMSIZE and truncate it if it is too long. If you want to be absolutely sure that no overflows occur, make each component area be FMSIZE bytes long.

SEE

strgfn, strmfe, strmfn

EXAMPLE

```
#include  <dos.h>
#include  <stdio.h>
#include  <stdlib.h>

char  a[3],b[FMSIZE],c[FNSIZE],d[FESIZE];

/*
 *  After  the  next  statement,  the  component  strings
 *  are:
 *
 *  a  =>  ""
 *  b  =>  "abc\\def"
 *  c  =>  "ghi"
 *  d  =>  ""
 */

 strsfn("abc\\def\\ghi",a,b,c,d);

/*
 *  After  the  next  statement,  the  component  strings
 *  are:
 *  a  =>  "b:"
 *  b  =>  ""
 *  c  =>  "myfile"
 *  d  =>  "str"
*/

 strsfn("b:myfile.str",a,b,c,d);
```

strspn, strcspn, stcis, stcisn

Measure character span

Class: ANSI *Category: String Search*

SYNOPSIS

```
#include  <string.h>

len  =  strspn(s,b);     Measure  span  of  chars  in  set
len  =  strcspn(s,b);    Measure  span  of  chars  not  in  set
len  =  stcis(s,b);      Measure  span  of  chars  in  set
len  =  stcisn(s,b);     Measure  span  of  chars  not  in  set

size_t  len;             span  length  in  bytes
const  char  *s;         points  to  string  being  scanned
const  char  *b;         points  to  character  set  string
```

DESCRIPTION

These functions measure the number of characters at the beginning of input string s that are either in or not in the character set specified by b. The stcis and strspn functions are identical and count the number of leading characters that are in the set. Similarly, stcisn and strcspn are identical and count the number of leading characters that are not in the set. The stc pair are provided for compatibility with other versions of Lattice C, while the str functions are now part of the ANSI standard.

RETURNS

The functions all return the number of bytes that are in or not in the specified character set. Note that the scan always stops when the null terminator byte is reached.

EXAMPLE

```
#include  <stdio.h>
#include  <string.h>
int  main(void)
{
  char  s1[256],s2[256];

  for(;;)  {
    printf("\nEnter  test  string:  ");
    if(gets(s1)  ==  NULL)  exit(0);
    printf("Enter  span  string:  ");
    if(gets(s2)  ==  NULL)  exit(0);
    printf("strspn:  %ld\n",(long)strspn(s1,s2));
    printf("strcspn:  %ld\n",(long)strcspn(s1,s2));
    printf("stcis:  %d\n",stcis(s1,s2));
    printf("stcisn:  %d\n",stcisn(s1,s2));
  }
  return  0;
}
```

strsrt — Sort string pointer list

Class: Lattice *Category: Search and Sort*

SYNOPSIS

```
#include  <string.h>

strsrt(s,n);

char  *s[];       string  pointer  list
size_t  n;        number  of  pointers  in  list
```

DESCRIPTION

This function performs a simple insertion sort of the string pointers in the specified list. It is particularly useful in conjunction with the getfnl and strbpl functions. For large lists, you will usually get better performance using tqsort.

SEE

getfnl, strbpl, tqsort

EXAMPLE

```
/*
 *  This  program  constructs  an  array  of  pointers  to
 *  all  file  names  in  the  current  directory  that  have
 *  a  ".c"  extension.  Then  the  array  is  sorted  by
 *  ASCII  order.
 */

#include  <stdlib.h>
#include  <string.h>

char  names[3000],*pointers[300];

void  foo(void)
{
  int  count;

  count  =  getfnl("*.c",names,sizeof(names),0);
  if(count  >  0)
  {
    if(strbpl(pointers,300,names)  !=  count)
      break;
  strsrt(pointers,count);
  }
}
```

strstr — Locate first occurrence of substring in string

Class: ANSI *Category: String Search*

SYNOPSIS

```
#include  <string.h>

ptr  =  strstr(s1,s2);

char  *ptr;          pointer  to  substring  in  string
const  char  *s1;    string  to  be  searched
const  char  *s2;    substring  to  locate
```

DESCRIPTION

The strstr function locates the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2.

RETURNS

The strstr function returns a pointer to the located string, or a NULL pointer if the string is not found. If s2 points to a string with zero length, the function returns s1.

strtod — Convert initial string portion to double

Class: ANSI *Category: Data Conversion/Formatting*

SYNOPSIS

```
#include  <stdlib.h>

double  strtod(const  char  *nptr,  char  **endptr);

double  val;          converted  value
const  char  *nptr;   string  portion  to  be  converted
char  **endptr;       points  to  object  containing
                      pointer  to  final  string
```

DESCRIPTION

The strtod function converts the initial portion of the string pointed to by nptr to double representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the isspace function), a subject sequence resembling a floating-point constant; and a final string of one or more unrecognised characters, including the terminating null character of the input string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.

The expected form of the subject sequence is an optional plus or minus sign, then a nonempty sequence of digits optionally containing a decimal-point character, then an optional exponent part, but *no* floating suffix. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign, a digit, or a decimal point character.

If the subject sequence has the expected form, the sequence of characters starting with the first digit or the decimal point character (whichever occurs first) is interpreted as a floating point constant, except that the decimal point character is used in place of a period, and that if neither an exponent part nor a decimal-point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a NULL pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a NULL pointer.

The strtod and strtol functions have been adopted by ANSI (from UNIX System V) because they offer more control over the conversion process, and because they are not required to produce unexpected results on overflow during conversion.

RETURNS

The strtod function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus HUGE_VAL is returned (according to the sign of the value), and the value of the macro ERANGE is stored in errno. If the correct value would cause underflow, zero is returned and the value of the macro ERANGE is stored in errno. Upon coverting the first part of nptr, the strtod function returns a pointer to the first character that is not part of the number. The converted double is returned.

strtok

Get a token

Class: ANSI *Category: String Search*

SYNOPSIS

```
#include <string.h>

t = strtok(s,b);

char  *t;          token  pointer
char  *s;          input  string  pointer  or  NULL
const  char  *b;   break  character  string  pointer
```

DESCRIPTION

This function treats the input string as a series of one or more tokens separated by one or more characters from the break string. By making a sequence of calls to strtok, you can obtain the tokens in left-to-right order. To get the first (leftmost) token, supply a non-NULL pointer for the s argument. Then to get the next tokens, call the function repeatedly with a NULL pointer for s, until you get a NULL return pointer to indicate that there are no more tokens. The break string can be changed from one call to another.

Each time it is entered, strtok takes the following steps:

- If the input string is NULL, obtain the string pointer that was used on the preceding call. Otherwise use the new input string pointer.
- Scan forward through the string to the next non-break character. If it is a null byte, return a value of NULL to indicate that there are no more tokens.
- Scan forward through the string to the next break character or the null terminator. In the former case, write a null byte into the string to terminate the token, and then scan forward until the next non-break is found. In either case, save the final value of the string pointer for the next call, and return the token pointer.

Note that the input string gets changed as the scan progresses. Specifically, a null byte is written at the end of each token.

RETURNS

A NULL pointer is returned when there are no more tokens.

SEE

stptok, strcspn, strspn

EXAMPLE

```
/*
 *
 *  This example breaks out words that are separated
 *  by blanks or commas. The token pointer takes on
 *  the following values as the program loops:
 *
 *  LOOP  TOKEN
 *  1     "first"
 *  2     "second"
 *  3     "third"
 *  4     "fourth"
 *  5     NULL
 */

#include  <string.h>
#include  <stdio.h>

int  main(void)
{
  char  test[]  =  "first,  second  third,  fourth";
  char  *token;

  token  =  strtok(test,",  ");
  while(token  !=  NULL)
  {
    printf("%s\n",token);
    token  =  strtok(NULL,",  ");
  }
  return  0;
}
```

strtol

Convert string to long integer

Class: ANSI

Category: Data Conversion/Formatting

SYNOPSIS

```
#include  <stdlib.h>

r  =  strtol(p,np,base);

long  int  r;       result
const  char  *p;    input  string  pointer
char  **np;         receives  new  input  string  pointer
int  base;          conversion  base
```

DESCRIPTION

This function converts an ASCII input string into a long integer according to the specified base, which can range from 0 to 36, excluding 1. Valid digit characters are 0 to 9, a to z, and A to Z. The highest allowable character is determined by the conversion base. For example, if the base is 17, then the string can contain digits from 0 to 9, a to g, and A to G.

The function skips leading white space and then checks for a leading plus or minus sign. In the latter case, the result of the conversion is negated before it is returned. The conversion stops at the first invalid character, and a pointer to that character is returned in np if the np argument is not NULL. Note that if the entire string is converted, np will contain a pointer to the null terminator byte.

If base is 0, the string is analysed to see if it is octal, decimal, or hexadecimal:

Base 16
If the string begins with 0x or 0X, base 16 (hexadecimal) conversion is performed.

Base 8
Otherwise, if the string begins with 0, base 8 (octal) conversion is performed.

Base 10
If neither of the above applies, base 10 (decimal) conversion is performed.

RETURNS

The strtol function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, LONG_MAX is returned for overflow, or LONG_MIN for underflow. The value of the macro ERANGE is stored in errno.

SEE

atol, stcd_l, strtoul

EXAMPLE

```
/*
 * This  program  tests  the  strtol  function.
 */

#include  <stdio.h>
#include  <string.h>

int  main(void)
{
  char  *p,buff[80];
  int  base;
  long  x;

  for  (;;)
  {
    printf("\nEnter  number  base  (0  to  36):  ");
    if(gets(buff)  ==  NULL)
      break;
    if(buff[0]  ==  '\0')
      break;
    base  =  atoi(buff);
    if((base  <  0)  ||  (base  >  36))
      continue;
    printf("Enter  number:  ");
    if(gets(buff)  ==  NULL)
      break;
    if(buff[0]  ==  '\0')  exit(0);
    x  =  strtol(buff,&p,base);
    printf("Decimal  result  =  %ld\n",x);
    if(*p  !=  '\0')
      printf("Residual  =  %s\n",p);
  }
  return  0;
}
```

strtoul   Convert initial string portion to unsigned long

Class: ANSI   *Category: Data Conversion/Formatting*

SYNOPSIS

```
#include  <stdlib.h>

val  =  strtoul(nptr,eptr,base);

unsigned long int val;    converted value
const char *nptr;         string portion to be
                          converted
int base;                 radix specifier
char **eptr;              points to object containing
                          pointer to final string
```

DESCRIPTION

The strtoul function converts the initial portion of the string pointed to by nptr to unsigned long int representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the isspace function), a subject sequence resembling an unsigned integer represented in some radix determined by base; and a final string of one or more unrecognised characters, including the terminating null character of the input string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of base is zero, the expected form of the subject sequence is that of an integer constant, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of base is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by base, optionally preceded by a plus or minus sign, but not including an integer suffix. The letters from a (or A) through z (or Z) are ascribed the values 10 to 35; only letters whose ascribed values are less than that of base are permitted. If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) is interpreted as an integer constant according to the ANSI syntax. If the subject sequence has the expected form and the value of base is between 2 and 37, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a NULL pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to endptr, provided that eptr is not a NULL pointer.

RETURNS

The strtoul function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ULONG_MAX is returned, and the value of the macro ERANGE is stored in errno.

SEE

atol, stcd_l, strtol

strxfrm — Transform string and place into array

Class: ANSI *Category: Localisation*

SYNOPSIS

```
#include <string.h>

len = strxfrm(s1,s2,n);

size_t len;       length of transformed string
char *s1;         array containing transformed string
const char *s2;   pointer to string to be transformed
size_t n;         maximum number of characters to
                  place
```

DESCRIPTION

The strxfrm function transforms the string pointed to by s2 and places the resulting string into the array pointed to by s1. The transformation is such that if the strcmp function is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the strcoll function applied to the same two original strings. No more than n characters are placed into the resulting array pointed to by s1, including the terminating null character. If n is zero, s1 is permitted to be a NULL pointer. If copying takes place between objects that overlap, the behaviour is undefined.

The strcoll and strxfrm functions provide for locale-specific string sorting. The strcoll function is intended for applications in which the number of comparisons is small, while strxfrm is more appropriate when items are to be compared a number of times; the cost of transformation is then only paid once.

RETURNS

The strxfrm function returns the length of the transformed string (not including the terminating null character). If the value returned is n or more, the contents of the array pointed to by s1 will not be null ('\0') terminated.

EXAMPLE

```
/*
 * The value of the following expression is the size
 * of the array needed to hold the transformation of
 * the string pointed to by s.
 */

size_t len(const char *s)
{
  return 1 + strxfrm(NULL, s, 0);
}
```

stspfp — Parse file path

Class: Lattice *Category: File Name Manipulation*

SYNOPSIS

```
#include <string.h>

error = stspfp(path,nx);

int error;       -1 for error, 0 for success
char *path;      file path string
int nx[16];      node index array
```

DESCRIPTION

This function parses a file path, which is a null-terminated string consisting of nodes separated by the _SLASH character. Each separator is replaced with a null byte, and the index to the first character of that node is placed into the node index array. The last entry in the array is followed by a -1. A leading separator in the path string is skipped.

RETURNS

A return value of -1 indicates that the path contains more than 15 nodes.

SEE

stcgfe, stcgfn, stcgfp, strsfn, _SLASH

EXAMPLE

```
/*
 * The following parses \ABC\DE\F into strings ABC,
 * DE, and F. The node index array will then contain
 * 1, 5, 8, and -1.
 */

#include <string.h>

int xx[16];

stspfp("\\ABC\\DE\\F",xx);
```

_stub

Default routine for undefined routines

Class: Lattice *Category: Process Environment*

SYNOPSIS

```
    _stub();
```

DESCRIPTION

The _stub function is the default routine resolved by CLink for routines not found in libraries. By default, it will give a prompt indicating that the unwritten routine had been called. It is intended to allow development and testing of a program for which some of the routines have not been written (and, of course, are not expected to be called).

swab

Byte swap words

Class: UNIX *Category: Data Conversion/Formatting*

SYNOPSIS

```
    #include  <stdlib.h>

    swab(src,dest,nbytes);

    const void *src;      area to copy bytes from
    void *dest;           area to copy bytes to
    size_t nbytes;        number of bytes to exchange
```

DESCRIPTION

The swab function copies nbytes from src to dest, exchanging odd and even bytes as it does so. The value of nbytes should be even, also note that this function is undefined in the general overlapping block case (cf. memcpy), however when src==dest the function will perform as expected.

Note that this function is most often used when transferring data from one architecture to another (e.g. Intel - Motorola), where the order of bytes within words differs.

SEE

memmove, memcpy

system — Call system command processor

Class: ANSI *Category: Process Creation*

SYNOPSIS

```
#include  <stdlib.h>

error  =  system(cmd);

int  error;                    non-zero  if  error
const  char  *cmd;             command  string

extern  char  *_comspecmagic;  "/c"
extern  char  *_shellmagic;    "-c"
```

DESCRIPTION

This function invokes the system command processor and passes the cmd string to it. The function will attempt to find a command processor by inspecting the _shell_p system variable, if this is non-NULL it will call through this vector with cmd as the sole argument.

If no resident shell can be found a command processor specified by SHELL or the COMSPEC environment variable is searched for, and so you must be sure that this variable is properly specified in your environment (if one is available). Under normal circumstances, you will automatically inherit a copy of this variable if your program starts. If neither of these exist (e.g. the program was run from the desktop), system will attempt to start a process using the forkl function.

When using the SHELL or COMSPEC environment variables many command processors require a command line switch to force them to accept a command on their command line, system makes the variables _shellmagic and _comspecmagic which have the default values "-c" and "/c" respectively. You may change these simply by moving the pointer to a new area of your own. Note that you should not copy into the old area as this has a strictly limited size.

If the cmd passed to system is NULL, then the return value specifies whether a command processor is available. Under GEMDOS this value will always be non-zero indicating that a command processor is available.

RETURNS

If the command processor cannot be invoked, a value of -1 is returned, and additional error information can be found in errno and _OSERR. Otherwise, the function returns the value that was passed back by the command processor.

SEE

errno, forkl, _OSERR

EXAMPLE

```
/*
 *  Run  all  the  programs  mentioned  on  the  command
 *  line  one  after  another
 */

#include  <stdlib.h>
#include  <stdio.h>

int  main(int  argc,  char  *argv[]);
{
  while  (--argc)
    system(*++argv);
  return  0;
}
```

time

Get system time in seconds

Class: ANSI *Category: Date and Time*

SYNOPSIS

```
    #include  <time.h>

    timeval  =  time(timeptr);

    time_t  timeval;  time value
    time_t  *timeptr; pointer to time value storage
```

DESCRIPTION

This function returns the current time expressed as the number of seconds since 00:00:00 Greenwich Mean Time, January 1, 1970. If timeptr is not NULL, the time value is also stored in that location.

SEE

asctime, ctime, gmtime, localtime, _tzset, utpack, utunpk

EXAMPLE

```
    #include  <time.h>
    #include  <stdio.h>

    int  main(void)
    {
      time_t  t;

      time(&t);
      printf("Current  time  is  %s\n",ctime(&t));

      return  0;
    }
```

_timedata

Time Zone variables

Class: UNIX *Category: Date and Time*

SYNOPSIS

```
    extern  int   __daylight;      Daylight  savings  time  flag
    extern  long  __timezone;      Timezone  bias  from  GMT
    extern  char  *__tzname[2];    Timezone  names
    extern  char  __tzstn[4];      Standard  time  name
    extern  char  __tzdtn[4];      Daylight  time  name
```

DESCRIPTION

These variables are initialised by the _tzset function and are used by the localtime function to adjust from Greenwich Mean Time (GMT) to the local time.

The __daylight item is non-zero if daylight saving time is currently in effect. The __timezone value is the number of seconds that must be subtracted from GMT. The two __tzname pointers point to __tzstn and __tzdtn, respectively. These strings contain the three-character names for standard time (__tzstn) and daylight time (__tzdtn).

SEE

localtime, _tzset

tmpfile

Create a temporary binary file

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
#include <stdio.h>

strm = tmpfile();

FILE *strm; pointer to file stream
```

DESCRIPTION

The tmpfile function creates a temporary binary file (mode "wb+") that will automatically be removed when it is closed or at program termination. If the program terminates abnormally the file may not be deleted correctly.

RETURNS

The tmpfile function returns a pointer to the stream of the file that it created. If the file cannot be created, the tmpfile function returns a NULL pointer.

SEE

fopen, mktemp, tmpnam

tmpnam

Create temporary file name

Class: ANSI

Category: Stream I/O

SYNOPSIS

```
#include <stdio.h>

name = tmpnam(buff);

char *name;          points to file name
char *buff;          buffer for file name or NULL
```

DESCRIPTION

This function creates a unique file name and returns a pointer to the name.

If buff is not NULL, then the file name is placed in that buffer, and name will be the same as buff. The buffer must be large enough to hold the file name; L_tmpnam (defined in stdio.h) is a safe size.

If buff is NULL, then an internal buffer is used, and the function returns a pointer to it. Note that this internal buffer is changed on every call to tmpnam, even if buff is not NULL.

In previous releases, the file was created when the unique name was selected. In accordance with the ANSI standard, this no longer done.

RETURNS

A NULL return indicates that the unique file could not be created.

to... — Character conversion functions

Class: ANSI *Category: Character Classification/Conversion*

SYNOPSIS

```
#include  <ctype.h>

cc  =  toascii(c);      convert  character  to  ASCII
cc  =  tolower(c);      convert  character  to  lower  case
cc  =  toupper(c);      convert  character  to  upper  case
cc  =  _tolower(c);     convert  upper  case  character  to
                        lower  case
cc  =  _toupper(c);     convert  lower  case  character  to
                        upper  case

int  cc;                converted  character
int  c;                 character  to  convert
```

DESCRIPTION

These functions convert characters into different forms. The toascii conversion simply resets all high-order bits, leaving only the lower seven. The tolower conversion tests if c is an upper case alphabetic character and, if so, converts it to lower case. Otherwise, cc is the same as c. The toupper conversion is the reverse of tolower.

The _toupper and _tolower functions perform a similar function to toupper and tolower, but do not check the case of the character before converting it. They are provided primarily for compatability with other systems and do *not* form part of the ANSI standard.

SEE

_ctype

EXAMPLE

```
/*
 *  Echoe  input  lines  in  upper  case.
 */
#include  <stdio.h>
#include  <ctype.h>
int  main(void)
{
  char  b[100],*p;

  while(gets(b)  !=  NULL)  {
    for(p  =  b;  *p  !=  '\0';  p++)
      *p  =  toupper(*p);
      puts(b);
  }
  return  0;
}
```

_tos — Operating system version number

Class: GEMDOS *Category: Process Environment*

SYNOPSIS

```
#include  <dos.h>

extern  short  _tos;    major  and  minor  OS  version
```

DESCRIPTION

These variable gives the operating system version number, in the major/minor form used in the ROM. The high eight bits give the major version number (1 on all current releases), whilst the lower eight bits give the minor release number.

The currently used values are:

Major	Minor	Name
1	0	ROM TOS (1.0)
1	2	Blitter TOS (1.2)
1	4	Rainbow TOS (1.4)
1	6	STE TOS (1.6)

SEE

Sversion

EXAMPLE

```
/*
 *  print  out  OS  version  number
 */

#include  <dos.h>
#include  <stdio.h>

void  show_version(void)
{
  printf("TOS  version=%d.%d\n",_tos>>8,_tos&0xff);
}
```

trig

Trigonometric functions

Class: ANSI *Category: Mathematics*

SYNOPSIS

```
    #include <math.h>

    r = cos(x);          Cosine function
    r = sin(x);          Sine function
    r = tan(x);          Tangent function

    r = acos(x);         Arccosine function
    r = asin(x);         Arcsine function
    r = atan(x);         Arctangent function
    r = atan2(x,y);      Arctangent of x/y

    r = cosh(x);         Hyperbolic cosine function
    r = sinh(x);         Hyperbolic sine function
    r = tanh(x);         Hyperbolic tangent function

    double r;            result;
    double x,y;          arguments
```

DESCRIPTION

The cos, sin, and tan routines compute the usual circular functions of angles expressed in radians.

The acos, asin, atan, and atan2 routines compute the inverse circular functions, returning angular values expressed in radians. Results are constrained as follows:

Function	Return Range
acos	0 to π
asin	$-\frac{\pi}{2}$ to $\frac{\pi}{2}$

Function	Return Range
atan	$-\frac{\pi}{2}$ to $\frac{\pi}{2}$
atan2	$-\frac{\pi}{2}$ to $\frac{\pi}{2}$

Since the tangent becomes very large for angles close to $\frac{\pi}{2}$, the atan2 function is often used to avoid computations with large numbers that might easily overflow. With atan2, you can express the large tangent value as a quotient of two more reasonable numbers.

The cosh, sinh, and tanh routines compute the normal hyperbolic functions.

SEE

matherr

_tzset

Set time zone variables

Class: XENIX *Category: Date and Time*

SYNOPSIS

```
    #include <time.h>

    _tzset();

    /* These symbols are defined in time.h:
     *
     * extern int  __daylight;
     * extern long __timezone;
     * extern char *__tzname[2];
     * extern char __tzstn[4];
     * extern char __tzdtn[4];
     */
```

DESCRIPTION

The _tzset function assigns values to the time zone variables __daylight, __timezone, and __tzname. These variables are then used by localtime and other functions to correct from Greenwich Mean Time (GMT) to local time.

The values for these variables are obtained from the environment variable named TZ having the following form

```
    set TZ=aaabbbccc
```

where aaa is the 3-letter abbreviation for the local standard time zone (e.g. CET), and bbb is a number from -23 to +24 indicating the value that is subtracted from GMT in order to obtain local standard time. Both aaa and bbb are required, but ccc is the abbreviation for the local daylight savings time zone (e.g. BST), and it should be present only if daylight savings time is currently in effect.

When _tzset is called, it first tries to locate TZ in the environment string array and uses the default string "GMT0" if TZ isn't found. Then __timezone is loaded with the number of seconds that must be subtracted from GMT in order to get the local time. Next __daylight is loaded with 0 if the ccc portion of TZ is absent and 1 if ccc is present. Then the aaa and ccc parts are copied to __tzstn and __tzdtn, respectively, with null terminators. Finally, __tzname(0) and __tzname(1) are loaded with pointers to __tzstn and __tzdtn respectively.

SEE

_timedata, localtime

ungetc — Push input character back

Class: ANSI *Category: Stream I/O*

SYNOPSIS

```
#include  <stdio.h>

r  =  ungetc(c,  fp);

int  r;          return  character  or  code
int  c;          character  to  be  pushed  back
FILE  *fp;       file  pointer
```

DESCRIPTION

This function pushes a character back to the specified buffered input file. The character need not be the same as the one that was most recently read. However, before calling ungetc, you must have read at least one character via fgetc or one of the other buffered input functions. Also, you can only push back one character; if you call ungetc more than once between input functions, the results are undefined.

RETURNS

Normally ungetc returns the character that was pushed back. However, if the end-of-file has been hit or if no characters have been read yet, the value EOF is returned.

SEE

fgetc, fgets, getc, gets

EXAMPLE

```
#include  <stdio.h>
#include  <ctype.h>
int  main(void)
{
  int  c;

  for  (;;)
  {
    printf("Loop  1...\n");
    while((c  =  getchar())  !=  EOF)
      if(isalpha(c))
        putchar(c);
      else
        break;
    ungetc(c);
  }
  printf("\n\nDone\n");
  return  0;
}
```

ungetch — Unget console keyboard character

Class: Lattice *Category: Console and Port I/O*

SYNOPSIS

```
#include  <dos.h>

r  =  ungetch(c);

int  c;
int  r;
```

DESCRIPTION

The ungetch function is one of a group of functions that perform I/O operations with the keyboard and display attached as the console device.

The ungetch function pushes a character onto a stack so that it will be read on the next call to getch or getche. Also, kbhit will report that a character is waiting if one has been pushed onto the stack. The stack is only one level deep, and if you try to push a second character, the function will return -1 and ignore the request. Otherwise, it returns the character that was pushed. Also, note that if you push back a non-ASCII scan code, the next call to getch or getche will not produce the usual zero return to indicate that a scan code is coming. You can clear the stack by calling ungetch with a character value of 0.

RETURNS

As noted above.

SEE

cgets, cputs, getch, getche, kbhit, putch

utime — Set file modification time

Class: UNIX *Category: Low-Level I/O*

SYNOPSIS

```
#include  <sys/types.h>
#include  <time.h>

err  =  utime(name,time)

int  err;                 error  return
const  char  *name;       name  of  file  to  manipulate
struct  utimbuf  *time;   time  buffer
```

DESCRIPTION

The utime function changes the last modified time of the file name. If the value of time is NULL, then the modification time is set to the current time, if it is not NULL then it should point to utimbuf structure which has the following elements:

```
struct  utimbuf
{
  time_t  actime;      /*  access  time  -  ignored  */
  time_t  modtime;     /*  new  last  modification  time  */
};
```

The modification time that is required should be placed in the modtime element of the structure.

RETURNS

The function returns 0 on succesfully changing the time of the file, or -1 to indicate an error, with further information in errno.

SEE

Fdatime, stat, time

utpack, utunpk — Pack or unpack UNIX time

Class: Lattice *Category: Date and Time*

SYNOPSIS

```
#include  <stdlib.h>

ut  =  utpack(x);      Pack  UNIX  time
utunpk(ut,x);          Unpack  UNIX  time

long  ut;              packed  UNIX  time
char  *x;              unpacked  UNIX  time
```

DESCRIPTION

These functions pack and unpack the 32-bit value time that is traditionally used in UNIX systems. This value is the number of seconds since 00:00:00, January 1, 1970. The time function returns the system clock in this form relative to Greenwich Mean Time.

The unpacked time is a 6-byte array in the following format:

Byte	Contents
x(0)	year - 1970 (-128 to +127)
x(1)	month (1 to 12)
x(2)	day (1 to 31)
x(3)	hour (0 to 23)
x(4)	minute (0 to 59)
x(5)	second (0 to 59)

Although this array is similar to the one produced by getclk and used by stpdate, note that the year is biased relative to 1970 instead of 1980. So, if you use utunpk followed by stpdate, you must subtract 10 from x(0) before the stpdate call. Note also that the year is a signed character and can be negative. A value of -3, for example, is 1967 (i.e. 1970 - 3).

SEE

ctime, getclk, gmtime, localtime, stpdate, time

EXAMPLE

```
/*
 * Get a file time and convert it to UNIX time.
 * No error checks.
 */

#include <time.h>
#include <dos.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
  char tt[6];
  int fh;
  long ft,ut;

  ft = getft(argv[1]);
  ftunpk(ft,tt);
  tt[0] += 10;
  ut = utpack(tt);
  printf("File time is: %s\n",ctime(&ut));
  return 0;
}
```

vfprintf

Formatted print to file, variable argument list

Class: ANSI *Category: Formatted I/O*

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

length = vfprintf(fp,fmt,arg);

int length;             number of characters generated
FILE *fp;               file pointer
const char *fmt;        format string
va_list arg;            variable argument list
```

DESCRIPTION

The vfprintf function is equivalent to fprintf, with the variable argument list replaced by arg, which has been initialised by the va_start macro (and possibly subsequent va_arg calls). The vfprintf function does not invoke the va_end macro.

RETURNS

The vfprintf function returns the number of characters transmitted, or a negative value if an output error occurred.

SEE

printf, vprintf, vsprintf

EXAMPLE

```
/*
 * generalised error handler
 */

#include <stdio.h>
#include <stdarg.h>

void error(const char *s,...)
{
  va_list args;

  fputs("Error: ",stderr);
  va_start(args, s);
  vfprintf(stderr, s, args);
  va_end(args);
  fputc('\n',stderr);
  exit(EXIT_FAILURE);
}
```

vprintf

Formatted print to stdout, variable argument list

Class: ANSI *Category: Formatted I/O*

SYNOPSIS

```
#include  <stdarg.h>
#include  <stdio.h>

length  =  vprintf(fmt,arg);

int  length;            number  of  characters  generated
const  char  *fmt;      format  string
va_list  arg;           variable  argument  list
```

DESCRIPTION

The vprintf function is equivalent to printf, with the variable argument list replaced by arg, which has been initialised by the va_start macro (and possibly subsequent va_arg calls). The vprintf function does not invoke the va_end macro.

RETURNS

The vprintf function returns the number of characters transmitted, or a negative value if an output error occurred.

SEE

printf, vfprintf, vsprintf

vsprintf

Formatted print to storage, variable argument list

Class: ANSI *Category: Formatted I/O*

SYNOPSIS

```
#include  <stdarg.h>
#include  <stdio.h>

length  =  vsprintf(s,fmt,arg);

int  length;            number  of  characters  generated
char  *s;               storage  pointer
const  char  *fmt;      format  string
va_list  arg;           variable  argument  list
```

DESCRIPTION

The vsprintf function is equivalent to sprintf, with the variable argument list replaced by arg, which has been initialised by the va_start macro (and possibly subsequent va_arg calls). The vsprintf function does not invoke the va_end macro. If copying takes place between objects that overlap, the behaviour is undefined.

RETURNS

The vsprintf function returns the number of characters written in the array, not counting the terminating null character.

SEE

printf, vfprintf, vprintf

wait

Wait for child process to complete

Class: UNIX *Category: Process Creation*

SYNOPSIS

```
    #include  <stdlib.h>

    cc  =  wait();

    int  cc;  completion  code
```

DESCRIPTION

The wait function is used in conjunction with the fork functions, which create a "child process" by loading a new program and passing control to it. When the child process completes, the current program (i.e., the parent process) can obtain its completion code via the wait function.

When a child process is created under GEMDOS, the parent suspends execution until the child is finished. The wait function must be called to obtain the child process's completion code.

RETURNS

If the specified program file cannot be found using the fork function, a -1 return is made, and additional error information can be found in errno and _OSERR. Note that you must call the wait function in order to obtain the completion code from the child process.

SEE

Pexec, exit, fork

wcstombs

Multibyte string conversion

Class: ANSI *Category: Wide Characters*

SYNOPSIS

```
    #include  <stdlib.h>

    num  =  wcstombs(s,pwcs,n);

    size_t  num;                  number  of  bytes  modified
    char  *s;                     string  to  be  converted
    const  wchar_t  *pwcs;        array  to  store  codes
    size_t  n;                    maximum  number  of  bytes  to  be
                                  modified
```

DESCRIPTION

The wcstombs function converts a sequence of codes that correspond to multibyte characters from the array pointed to by pwcs into a sequence of multibyte characters that begins in the initial shift state. It then stores these multibyte characters into the array pointed to by s, stopping if a multibyte character would exceed the limit of n total bytes or if a null character is stored. Each code is converted as if by a call to the wctomb function, except that the shift state of the wctomb function is not affected.

No more than n bytes will be modified in the array pointed to by s. If copying takes place between objects that overlap, the behaviour is undefined.

RETURNS

If a code is encountered that does not correspond to a valid multibyte character, the wcstombs function returns ((size_t)-1). Otherwise the wcstombs function returns the number of bytes modified, not including a terminating null character, if any.

SEE

wctomb

wctomb Determine bytes needed to represent multibyte character

Class: ANSI *Category: Wide Characters*

SYNOPSIS

```
    #include  <stdlib.h>

    ret  =  wctomb(s,wchar);

    int  ret;
    char  *s;           array  object  in  which  character  is
                        stored
    wchar_t  wchar;     multibyte  character  code  value
```

DESCRIPTION

The wctomb function determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is wchar (including any change in shift state). It stores the multibyte character representation in the array object pointed to by s (if s is not a NULL pointer). At most MB_CUR_MAX characters are stored. If the value of wchar is zero, the wctomb function is left in the initial shift state.

RETURNS

If s is a NULL pointer, the wctomb function returns a non-zero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings. If s is not a NULL pointer, the wctomb function returns -1 if the value of wchar does not correspond to a valid multibyte character, or returns the number of bytes that comprise the multibyte character corresponding to the value of wchar.

In no case will the value returned be greater than the value of the MB_CUR_MAX macro.

SEE

wcstombs

_xcovf Stack overflow exit

Class: GEMDOS *Category: Errors*

SYNOPSIS

```
    _xcovf();
```

DESCRIPTION

This error exit is called whenever a potential stack overflow is detected by the function prologue. In other words, if the stack does not contain enough space to handle the needs of a function, _xcovf will be called when that function is activated. The default version prints a stack overflow message on the screen and aborts with exit code 3. We supply the source code for this version so you can change it for your particular application.

Note that any user supplied function must be compiled with stack checks off, otherwise the function will recursively call itself!

SEE

_base, _STACK, _STKDELTA

Index