INTRODUCTION TO THE C STANDARD LIBRARY

The Standard Library contains functions which fall into two classes: high level I/O and convenience.

The high level I/O functions provide facilities that are normally considered part of the definition of other languages; for example, the FORMAT "statement" of Fortran. In addition, automatic buffering of I/O channels improves the speed of file access because fewer system calls are necessary.

The high level I/O functions should not be confused with the low level system calls with similar names. Nor should "file pointers" be confused with "path numbers". The standard library functions maintain a structure for each file open that holds status information and a pointer into the files buffer. A user program uses a pointer to this structure as the "identity" of the file (which is provided by "fopen()"), and passes it to the various I/O functions. The I/O functions will make the low level system calls when necessary.

USING A FILE POINTER IN A SYSTEM CALL, OR A PATH NUMBER IN A STANDARD LIBRARY CALL, is a common mistake among beginners to C and, if made, will be sure to CRASH YOUR PROGRAM.

The convenience functions include facilities for copying, comparing, and concatenating strings, converting numbers to strings, and doing the extra work in accessing system information such as the time.

In the pages which follow, the functions available are described in terms of what they do and the parameters they expect. The "USAGE" section shows the name of the function and the type returned (if not int). The declaration of arguments are shown as they would be written in the function definition to indicate the types expected by the function. If it is necessary to include a file before the function can be used, it is shown in the "USAGE" section by "#include <filename>".

Most of the header files that are required to be included, must reside in the "DEFS" directory on the default system drive. If the file is included in the source program using angle bracket delimiters instead of the usual double quotes, the compiler will append this path name to the file name. For example, "#include <stdio.h>" is equivalent to "#include /d0/defs/stdio.h", if "/d0" is the path name of the default system drive.

PLEASE NOTE that if the type of the value returned by a function is not INT, you should make a predeclaration in your program before calling it. For example, if you wish to use "atof()", you should predeclare by having "double atof();" somewhere in your program before a call to it. Some functions which have associated header files in the DEFS directory that should be

included, will be pre-declared for you in the header. An example of this is "ftell()" which is predeclared in "stdio.h". If you are in any doubt, read the header file.

Atof,Atoi,Atol - ASCII to number conversions

USAGE

```
double atof(ptr)
char *ptr;

long atol(ptr)
char *ptr;

int atoi(ptr)
char *ptr;
```

DESCRIPTION

Conversions of the string pointed to by "ptr" to the relevant number type are carried out by these functions. They cease to convert a number when the first unrecognized character is encountered.

Each skips leading spaces and tab characters. Atof() recognizes an optional sign followed by a digit string that could possibly contain a decimal point, then an optional "e" or "E", an optional sign and a digit string. Atol() and atoi() recognize an optional sign and a digit string.

CAVEATS

Overflow causes unpredictable results. There are no error indications.

Fflush,Fclose - flush or close a file

USAGE

```
#include <stdio.h>

fflush(fp)
FILE *fp;

fclose(fp)
FILE *fp;
```

DESCRIPTION

Fflush causes a buffer associated with the file pointer "fp" to be cleared by writing out to the file; of course, only if the file was opened for write or update. It is not normally necessary to call fflush, but it can be useful when, for example, normal output is to "stdout", and it is wished to send something to "stderr" which is unbuffered. If fflush were not used and "stdout" referred to the terminal, the "stderr" message will appear before large chunks of the "stdout" message even though the latter was written first.

Fclose calls fflush to clear out the buffer associated with "fp", closes the file, and frees the buffer for use by another fopen call.

The exit() system call and normal termination of a program causes fclose to be called for each open file.

SEE ALSO

System call close(), fopen(), setbuf().

DIAGNOSTICS

EOF is returned if "fp" does not refer to an output file or there is an error on writing to the file.

Feof,Ferror,Clearerr,Fileno - return status information of files

USAGE

```
#include <stdio.h>

feof(fp)
FILE *fp;

ferror(fp)
FILE *fp;

clearerr(fp)
FILE *fp;

fileno(fp)
FILE *fp;
```

DESCRIPTION

Feof returns non-zero if the file associated with "fp" has reached its end. Zero is returned on error.

Ferror returns non-zero if an error condition occurs on access to the file "fp"; zero is returned otherwise. The error condition persists, preventing further access to the file by other Standard Library functions, until the file is closed, or it is cleared by clearerr.

Clearerr resets the error condition on the file "fp". This does NOT "fix" the file or prevent the error from occurring again; it merely allows Standard Library functions at least to try.

CAVEATS

These functions are actually macros that are defined in "<stdio.h>" so their names cannot be redeclared.

SEE ALSO

System call open(), fopen().

Findstr,Findnstr - string search

USAGE

```
findstr(pos,string,pattern)
char *string,*pattern;

findnstr(pos,string,pattern,size)
char *string,*pattern;
```

DESCRIPTION

These functions search the string pointed to by "string" for the first instance of the pattern pointed to by "pattern" starting at position "pos" (where the first position is 1 not 0). The returned value is the position of the first matched character of the pattern in the string or zero if a match is not found.

Findstr stops searching the string when a null byte is found in "string".

Findnstr only stops searching at position "pos" + "len" so it may continue past null bytes.

CAVEATS

The current implementation does not use the most efficient algorithm for pattern matching so that use on very long strings is likely to be somewhat slower than it might be.

SEE ALSO

index(),rindex.

Fopen - open a file and return a file pointer

USAGE

```
#include <stdio.h>

FILE *fopen(filename,action)
char *filename,*action;

FILE *freopen(filename,action,streak)
char *filename,*action;
FILE *stream;

FILE *fdopen(filedes,action)
int filedes;
char *action;
```

DESCRIPTION

Fopen returns a pointer to a file structure (file pointer) if the file named in the string pointed to by "filename" can be validly opened with the action in the string pointed to by "action".

The valid actions are:

"r"	open for reading
"w"	create for writing
"a"	append(write) at end of file, or create for writing
"r+"	open for update
"w+"	create for update
"a+"	create or open for update at end of file
"d"	directory read

Any action may have an "x" after the initial letter which indicates to "fopen()" that it should look in the current execution directory if a full path name is not given, and the x also specifies that the file should have execute permission.

E.g. f = fopen("fred","wx");

Opening for write will perform a "creat()". If a file with the same name exists when the file is opened for write, it will be truncated to zero length. Append means open for write and position to the end of the file. Writes to the file via "putc()" etc. will extend the file. Only if the file does not already exist will it be created.

NOTE that the type of a file structure is pre-defined in "stdio.h" as FILE, so that a user program may declare or define a file pointer by, for example, FILE *f;

Three file pointers are available and can be considered open the moment the program runs:

stdin	the standard input - equivalent to path number 0
stdout	the standard output - equivalent to path number 1
stderr	standard error output- equivalent to path number 2

All files are automatically buffered except stderr, unless a file is made unbuffered by a call to setbuf() (q.v.).

Freopen is usually used to attach stdin, stdout, and stderr to specified files. Freopen substitutes the file passed to it instead of the open stream. The original stream is closed. NOTE that the original stream will be closed even if the open does not succeed.

Fdopen associates a stream with a file descriptor. The streams type(r,w,a) must be the same as the mode of the open file.

CAVEATS

The "action" passed as an argument to fopen must be a pointer to a string, NOT a character. For example

```
fp = fopen("fred","r"); is correct but
fp = fopen("fred",'r'); is not.
```

DIAGNOSTICS

Fopen returns NULL (0) if the call was unsuccessful.

SEE ALSO

System call open() Fclose()

Fread,Fwrite - read/write binary data

USAGE

```
#include <stdio.h>

fread(ptr, size, number, fp)
FILE *fp;

fwrite(ptr, size, number, fp)
FILE *fp;
```

DESCRIPTION

Fread reads from the file pointed to by "fp". "Number" is the number of items of size "size" that are to be read starting at "ptr". The best way to pass the argument "size" to fread is by using "sizeof". Fread returns the number of items actually read.

Fwrite writes to the file pointed to by "fp". "Number" is the number of items of size "size" reading them from memory starting at "ptr".

DIAGNOSTICS

Both functions return 0(NULL) at end of file or error.

SEE ALSO

System calls read(),write(). Fopen(),getc(),putc(),printf().

Fseek,Rewind,Ftell - position in a file or report current position

USAGE

```
#include <stdio.h>

fseek(fp, offset, place)
FILE *fp;
long offset;

rewind(fp)
FILE *fp;

long ftell(fp)
FILE *fp;
```

DESCRIPTION

Fseek repositions the next character position of a file for either read or write. The new position is at "offset" bytes from the beginning of the file if "place" is 0, the current position if 1, or the end if 2. Fseek sorts out the special problems of buffering.

NOTE that using "lseek()" on a buffered file will produce unpredictable results.

Rewind is equivalent to "fseek(fp,0l,0)".

Ftell returns the current position, measured in bytes, from the beginning of the file pointed to by "fp".

DIAGNOSTICS

Fseek returns -1 if the call is invalid.

SEE ALSO

System call lseek().

NOTE:
FSEEK(FP, 0L, 0)
LONG CONSTANT
0L or 0L

Getc,Getchar - return next character to be read from a file

USAGE

```
#include <stdio.h>

int getc(fp)
FILE *fp;

int getchar()

int getw(fp)
FILE *fp;
```

DESCRIPTION

Getc returns the next character from the file pointed to by "fp".

Getchar is equivalent to "getc(stdin)".

Getw returns the next two bytes from the file as an integer.

Under OS-9 there is a choice of service requests to use when reading from a file. "Read()" will get characters up to a specified number in "raw" mode i.e. no editing will take place on the input stream and the characters will appear to the program exactly as in the file. "Readln()", on the other hand, will honor the various mappings of characters associated with a Serial Character device such as a terminal and in any case will return to the caller as soon as a carriage return is seen on the input.

In the vast majority of cases, it is preferable to use "readln()" for accessing Serial Character devices and "read()" for any other file input. "Getc()" uses this strategy and, as all file input using the Standard Library functions is routed through "getc()", so do all the other input functions. The choice is made when the first call to "getc()" is made after the file has been opened. The system is consulted for the status of the file and a flag bit is set in the file structure accordingly. The choice may be forced by the programmer by setting the relevant bit before a call to "getc()". The flag bits are defined in "<stdio.h>" as "_SCF" and "_RBF" and the method is as follows: assuming that the file pointer for the file, as returned by "fopen()" is f,

```
f->_flag |= _SCF;
```

will force the use of "readln()" on input and

```
f->_flag |= _RBF;
```

will force the use of "read()". This trick may be played on the standard streams "stdin", "stdout" and "stderr" without the need for calling "fopen()" but before any input is requested from the stream.

DIAGNOSTICS

EOF(-1) is returned for end of file or error.

SEE ALSO

Putc(),fread(),fopen(),gets(),ungetc().

Gets, Fgets - input a string

USAGE

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s,n,fp)
char *s;
FILE *fp;
```

DESCRIPTION

Fgets reads characters from the file "fp" and places them in the buffer pointed to by "s" up to a carriage return('\n') but not more than "n" - 1 characters. A null character is appended to the end of the string.

Gets is similar to fgets, but gets is applied to "stdin" and no maximum is stipulated and the '\n' is replaced by a null.

Both functions return their first arguments.

CAVEATS

The different treatment of the "n" by these functions is retained here for portability reasons.

DIAGNOSTICS

Both functions return NULL on end-of-file or error.

SEE ALSO

Puts(),getc(),scanf(),fread().

Isalpha,Isupper,Islower,Isdigit,Isalnum,Isspace,Ispunct,Isprint, Iscntrl,Isascii - character classification

USAGE

```
#include <ctype.h>

isalpha(c)

etc.
```

DESCRIPTION

These functions use table look-up to classify characters according to their ascii value. The header file defines them as macros which means that they are implemented as fast, in-line code rather than subroutines.

Each results in non-zero for true or zero for false.

The correct value is guaranteed for all integer values in isascii, but the result is unpredictable in the others if the argument is outside the range -1 to 127.

The truth tested by each function is as follows:

isalpha	c is a letter
isdigit	c is a digit
isupper	c is an upper case letter
islower	c is a lower case letter
isalnum	c is a letter or a digit
isspace	c is a space, tab character, newline, carriage return or formfeed
iscntrl	c is a control character (0 to 32) or DEL (127)
ispunct	c is neither control nor alpha-numeric
isprint	c is printable (32 to 126)
isascii	c is in the range -1 to 127

L3tol,Ltol3 - convert between long integers and 3-byte integers

USAGE

```
l3tol(lp,cp,n)
long *lp;
char *cp;

ltol3(cp,lp,n)
long *lp;
char *cp;
```

DESCRIPTION

Certain system values, such as disc addresses, are maintained in three-byte form rather than four-byte; these functions enable arithmetic to be used on them.

L3tol converts a vector of "n" three-byte integers pointed to by "cp", into a vector of long integers starting at "lp".

Ltol3 does the opposite.

Longjmp,Setjmp - jump to another function

USAGE

```
include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env,val)
jmp_buf env;
```

DESCRIPTION

These functions allow the return of program control directly to a higher level function. They are most useful when dealing with errors and interrupts encountered in a low level routine.

"Goto" in C has scope only in the function in which it is used; i.e. the label which is the object of a "goto" may only be in the same function. Control can only be transferred elsewhere by means of the function call, which, of course, returns to the caller. In certain abnormal situations a programmer would prefer to be able to start some section of code again, but this would mean returning up a ladder of function calls with error indications all the way.

Setjmp is used to "mark" a point in the program where a subsequent longjmp can reach. It places in the buffer, defined in the header file, enough information for longjmp to restore the environment to that existing at the relevant call to setjmp.

Longjmp is called with the environment buffer as an argument and also, a value which can be used by the caller of setjmp as, perhaps, an error status.

To set the system up, a function will call setjmp to set up the buffer, and if the returned value is zero, the program will know that the call was the "first time through". If, however, the returned value is non-zero, it must be a longjmp returning from some deeper level of the program.

NOTE that the function calling setjmp must NOT HAVE RETURNED at the time of calling longjmp, and the environment buffer must be declared GLOBALLY.

Malloc,Free,Calloc - memory allocation

USAGE

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *calloc(nel,elsize)
unsigned nel,elsize;
```

DESCRIPTION

Malloc returns a pointer to a block of at least "size" free bytes.

Free requires a pointer to a block that has been allocated by malloc; it frees the space to be allocated again.

Calloc allocates space for an array. Nel is the number of elements in the array, and elsize is the size of each element. Calloc initializes the space to zero.

DIAGNOSTICS

Malloc, free, and calloc return NULL(0) if no free memory can be found or if there was an error.

Mktemp - create unique temporary file name

USAGE

```
char *mktemp(name)
char *name;
```

DESCRIPTION

Mktemp may be used to ensure that the name of a temporary file is unique in the system and does not clash with any other file name.

"Name" must point to a string whose last five characters are "X"; the Xs will be replaced with the ascii representation of the task id.

For example, if "name" points to "foo.XXXXX", and the task id is 351, the returned value points at the same place, but it now holds "foo.351".

SEE ALSO

System call "getpid()".

Printf,Fprintf,Sprintf - formatted output

USAGE

```
#include <stdio.h>

printf(control [,arg0[,arg1..]])
char *control;

fprintf(fp, control [,arg0[,arg1..]])
FILE *fp;
char *control;

sprintf(string,control [,arg0[,arg1..]])
string [];
char *control;
```

DESCRIPTION

These three functions are used to place numbers and strings on the output in formatted, human readable form.

Fprintf places its output on the file "fp", printf on the standard output, and sprintf in the buffer pointed to by "string". NOTE that it is the user's responsibility to ensure that this buffer is large enough.

The "control" string determines the format, type, and number of the following arguments expected by the function. If the control does not match the arguments correctly, the results are unpredictable.

The control may contain characters to be copied directly to the output and/or format specifications. Each format specification causes the function to take the next successive argument for output.

A format specification consists of a "%" character followed by (in this order) :

An optional minus sign ("-") that means left justification in the field.

An optional string of digits indicating the field width required. The field will be at least this wide and may be wider if the conversion requires it. The field will be padded on the left unless the above minus sign is present, in which case it will be padded on the right. The padding character is, by default, a space, but if the digit string starts with a zero ("0"), it will be "0".

An optional dot (".") and a digit string, the precision, which for floating point arguments indicates the number of digits to follow the decimal point on conversion, and for strings, the maximum number of characters from the string argument are to be printed.

An optional character "l" indicates that the following "d","x", or "o" is the specification of a long integer argument. NOTE that in order for the printing of long integers to take place, the source code must have in it somewhere the statement pflinit(), which causes routines to be linked from the library.

A conversion character which shows the type of the argument and the desired conversion. The recognized conversion characters are:

d,o,x,X	The argument is an integer and the conversion is to decimal, octal, or hexadecimal, respectively. "X" prints hex and alpha in upper case.
u	The argument is an integer and the conversion is to an unsigned decimal in the range 0 to 65535.
f	The argument is a double, and the form of the conversion is "[-]nnn.nnn". Where the digits after the decimal point are specified as above. If not specified, the precision defaults to six digits. If the precision is 0, no decimal point or following digits are printed.
e,E	The argument is a double and the form of the conversion is "[-]n.nnne(+or-)nn"; one digit before the decimal point, and the precision controls the number following. "E" prints the "e" in upper case.
g,G	The argument is a double, and either the "f" format or the "e" format is chosen, whichever is the shortest. If the "G" format is used, the "e" is printed in upper case.

NOTE in each of the above double conversions, the last digit is rounded.

ALSO NOTE that in order for the printing of floats or doubles to take place, the source program MUST have the statement pffinit() somewhere.

c	The argument is a character.
s	The argument is a pointer to a string. Characters from the string are printed up to a null character, or until the number of characters indicated by the precision have been printed. If the precision is 0 or missing, the characters are not counted.
%	No argument corresponding; "%" is printed.

SEE ALSO

Kernighan & Ritchie pages 145-147. Putc(),scanf().

Putc,Putchar,Putw - put character or word in a file

USAGE

```
#include <stdio.h>

char putc(ch,fp)
char ch;
FILE *fp;

char putchar(ch)
char *ch;

putw(n,fp)
FILE *fp;
```

DESCRIPTION

Putc adds the character "ch" to the file "fp" at the current writing position and advances the position pointer.

Putchar is implemented as a macro (defined in the header file) and is equivalent to "putc(ch,stdout)".

Putw adds the (two byte) machine word "n" to the file "fp" in the manner of putc.

Output via putc is normally buffered except
(a) when the buffering is disabled by "setbuf()", and
(b) the standard error output is always unbuffered.

DIAGNOSTICS

Putc and putchar return the character argument from a successful call, and EOF on end-of-file or error.

SEE ALSO

Fopen(),fclose(),fflush(),getc(),puts(),printf(),fread().

Puts,Fputs - put a string on a file

USAGE

```
#include <stdio.h>

puts(s)
char *s;

fputs(s,fp)
char *s;
FILE *fp;
```

DESCRIPTION

Fputs copies the (null-terminated) string pointed to by "s" onto the file "fp".

Puts copies the string "s" onto the standard output and appends "\n".

The terminating null is not copied by either function.

CAVEATS

The inconsistency of the new-line being appended by puts and not by fputs is dictated by history and the desire for compatibility.

Qsort - quick sort

USAGE

```
qsort(base,n,size,compfunc)
char *base;
int (*compfunc)(); /* which means: a pointer to a function
returning an int */
```

DESCRIPTION

Qsort implements the quick-sort algorithm for sorting an arbitrary array of items.

"Base" is the address of the array of "n" items of size "size". "Compfunc" is a pointer to a comparison routine supplied by the user. It will be called by qsort with two pointers to items in the array for comparison and should return an integer which is less than, equal to, or greater than 0 where, respectively, the first item is less than, equal to, or greater than the second.

Scanf,Fscanf,Sscanf - input string interpretation

USAGE

```
#include <stdio.h>

fscanf(fp,control,[,pointer...])
FILE *fp;
char *control;

scanf(control[,pointer...])
char *control;

sscanf(string,control[,pointer...])
char *string,*control;
```

DESCRIPTION

These functions perform the complement to "printf()" etc.

Fscanf performs conversions from the file "fp", scanf from the standard input, and sscanf from the string pointed to by "string".

Each function expects a control string containing conversion specifications, and zero or more pointers to objects into which the converted values are stored.

The control string may contain three types of fields:

(a) Spaces, tab characters, or "\n" which match any of the three in the input.

(b) Characters not among the above and not "%" which must match characters in the input.

(c) A "%" followed by an optional "*" indicates suppression of assignment, an optional field width maximum and a conversion character indicating the type expected.

A conversion character controls the conversion to be applied to the next field and indicates the type of the corresponding pointer argument. A field consists of consecutive non-space characters and ends at either a character inappropriate for the conversion or when a specified field width is exhausted. When one field is finished, white-space characters are passed over until the next field is found.

The following conversion characters are recognized :

d A decimal string is to be converted to an integer.

o An octal string; the corresponding argument should point to an integer.

x A hexadecimal string for conversion to an integer.

s A string of non-space characters is expected and will be copied to the buffer pointed to by the

corresponding argument and a null ("\0") appended. The user must ensure that the buffer is large enough. The input string is considered terminated by a space, tab or ("\n").

c A character is expected and is copied into the byte pointed to by the argument. The white-space skipping is suppressed for this conversion. If a field width is given, the argument is assumed to point to a character array and the number of characters indicated is copied to it. NOTE to ensure that the next non-white-space character is read use "%1s" and that TWO bytes are pointed to by the argument.

e,f A floating point representation is expected on the input and the argument must be a pointer to a float. Any of the usual ways of writing floating point numbers are recognized.

[This denotes the start of a set of match characters; the inclusion or exclusion of which delimits the input field. The white-space skipping is suppressed. The corresponding argument should be a pointer to a character array. If the first character in the match string is not "^", characters are copied from the input as long as they can be found in the match string, if the first character is the copying continues while characters cannot be found in the match string. The match string is delimited by a "]".

D,O,X Similar to d,o,x above, but the corresponding argument is considered to point to a long integer.

E,F Similar to e,f above, but the corresponding argument should point to a double.

% A match for "%" is sought; no conversion takes place.

Each of these functions returns a count of the number of fields successfully matched and assigned.

CAVEATS

The returned count of matches/assignments does not include character matches and assignments suppressed by "*". The arguments must ALL be pointers. It is a common error to call scanf with the value of an item rather than a pointer to it.

DIAGNOSTICS

These functions return EOF on end of input or error and a count which is shorter than expected for unexpected or unmatched items.

SEE ALSO

Atoi(),Atof(),Getc(),Printf() Kernighan and Ritchie pp 147-150

Setbuf - fix file buffer

USAGE

```
#include <stdio.h>

setbuf(fp,buffer)
FILE *fp;
char *buffer;
```

DESCRIPTION

When the first character is written to or read from a file after it has been opened by "fopen()", a buffer is obtained from the system if required and assigned to it. Setbuf may be used to forestall this by assigning a user buffer to the file.

Setbuf must be used after the file has been opened and before any I/O has taken place.

The buffer must be of sufficient size and a value for a manifest constant, BUFSIZ, is defined in the header file for use in declarations.

If the "buffer" argument is NULL (0), the file becomes unbuffered and characters are read or written singly.

NOTE that the standard error output is unbuffered and the standard output is buffered.

SEE ALSO

fopen(),getc(),putc().

Sleep - stop execution for a time

USAGE

```
sleep(seconds)
unsigned seconds;
```

DESCRIPTION

The current task is stopped for the specified time.
If "seconds" is zero, the task will sleep for one tick.

Strcat, Strncat, Strcmp, Strncmp, Strcpy, Strhcpy, Strncpy, Strlen,Index, Rindex- string functions

USAGE

```
char *strcat(s1,s2)
char *s1,*s2;

char *strncat(s1,s2,n)
char *s1,*s2;
int n;

strcmp(s1,s2)
char *s1,*s2;

char *strhcpy(s1,s2)
char *s1,*s2;

strncmp(s1,s2,n)
char *s1,*s2;
int n;

char *strcpy(s1,s2)
char *s1,*s2;

char *strncpy(s1,s2,n)
char *s1,*s2;
int n;

strlen(s)
char *s;

char *index(s,ch)
char *s,ch;

char *rindex(s,ch)
char *s,ch;
```

DESCRIPTION

All strings passed to these functions are assumed null-terminated.

Strcat appends a copy of the string pointed to by "s2" to the end of the string pointed to by "s1". Strncat copies at most "n" characters. Both return the first argument.

Strcmp compares strings "s1" and "s2" for lexicographic order and returns an integer less than, equal to or greater than 0 where, respectively, "s1" is less than, equal to or greater than "s2". Strncmp compares at most "n" characters.

Strcpy copies characters from "s2" to the space pointed to by "s1" up to and including the null byte. Strncpy copies exactly "n" characters. If the string "s2" is too short, the "s1" will be padded with null bytes to make up the difference. If "s2" is too long, "s1" may not be null-terminated. Both functions return the first argument.

Strhcpy copies string with sign bit terminator.

Strlen returns the number of non-null characters in "s".

Index returns a pointer to the first occurrence of "ch" in "s" or NULL if not found.

Rindex returns a pointer to the last occurrence of "ch" in "s" or NULL if not found.

CAVEATS

Strcat and strcpy have no means of checking that the space provided is large enough. It is the user's responsibility to ensure that string space does not overflow.

SEE ALSO

Findstr().

System - shell command request

USAGE

```
system(string)
char *string;
```

DESCRIPTION

System passes its argument to "shell" which executes it as a command line. The task is suspended until the shell command is completed and system returns the shell's exit status. The maximun length of string is 80 characters. If a longer string is needed, use os9fork.

SEE ALSO

System calls "os9fork()","wait()".

Toupper,Tolower - character translation

USAGE

```
#include <ctype.h>

int toupper(c)
int c;

etc.

int _toupper(c)
int c;

int _tolower(c)
int c;
```

DESCRIPTION

The functions toupper and tolower have as their domain the integers in the range -1 through 255. Toupper converts lower-case to upper-case, and tolower converts upper-case to lower-case. All other arguments are returned unchanged.

The macros _toupper and _tolower do the same things as the corresponding functions, but they have restricted domains and they are faster. The argument to _toupper must be lower-case, and the argument to _tolower must be upper-case. Arguments that are outside each macros domain, such as passing a lower-case to _tolower, yield garbage results.

Ungetc - put character back on input

USAGE

```
#include <stdio.h>

ungetc(ch,fp)
char ch;
FILE *fp;
```

DESCRIPTION

This function alters the state of the input file buffer such that the next call of "getc()" returns "ch".

Only one character may be pushed back, and at least one character must have been read from the file before a call to ungetc.

"Fseek()" erases any pushback.

DIAGNOSTICS

Ungetc returns its character argument unless no pushback could occur, in which case EOF is returned.

SEE ALSO

getc(),fseek()

WARNING!!!
GETC HAS
INT VARIABLE
UNGETC HAS
CHAR