NEWIO(III) NEWIO(III)

NAME

newio – a new IO subroutine package

INTRODUCTION

This new package of IO subroutines is recommended for preferred usage. It is intended that this package replace the Portable Library. Although it is not directly compatible, as discussed below, it is sufficiently similar that a set of relatively small, inexpensive adaptor routines exist which make it appear identical to the current Portable Library except in some very obscure details. The most crucial difference between this package and the Portable Library is that the current offering names streams in terms of pointers rather than by the integers known as 'file descriptors.' Thus, for example, the routine which opens a named file returns a pointer to a certain structure rather than a number; the routine which reads an open file takes as an argument the pointer returned from the open call.

GENERAL USAGE

Each program using the library must have the line

#include <stdio.h>

which defines certain macros and variables. The command to compile is

cc . . . -lS

All names in the include file intended only for internal use begin with an underscore '_' to reduce the possibility of collision with a user name. The names intended to be visible outside the package are

stdin	The stream pointer to the standard input file
stdout	The stream pointer to the standard output file
stderr	The stream pointer to the standard error file
EOF	is actually −1, and is the value returned by the read routines on end-of-file or error.
NULL	is a notation for the null pointer, returned by pointer-valued functions to indicate an error
FILE	expands to 'struct _iob' and is a useful shorthand when declaring pointers to streams.
BUFSIZ	is a number (viz. 512) of the size suitable for an IO buffer supplied by the user. See *setbuf*, below.

getc, getchar, putc, putchar, feof, ferror, fileno are defined as macros. Their actions are described below; they are mentioned here to point out that it is not possible to redeclare them and that they are not actually functions; thus, for example, they may not have breakpoints set on them.

The routines in this package, like the current Portable Library, offer the convenience of automatic buffer allocation and output flushing where appropriate. Absent, however, is the facility of changing the default input and output streams by assigning to 'cin' and 'cout.' The names 'stdin,' stdout,' and 'stderr' are in effect constants and may not be assigned to.

CALLS

The routines in the library are in nearly one-to-one correspondence with those in the Portable Library. In several cases the name has been changed. This is an attempt to reduce confusion. If the attempt is judged to fail the names may be made identical even though the arguments may be different. The order of this list generally follows the order used in the Portable Library document.

*FILE *fopen(filename, type)*

char *filename, *type;

Fopen opens the file and, if needed, allocates a buffer for it. *Filename* is a character string speci-

fying the name. *Type* is a character string (not a single character). It may be '"r",' '"w",' or '"a"' to indicate intent to read, write, or append. The value returned is a file pointer. If it is null the attempt to open failed.

*FILE *freopen(filename, type, ioptr)*
char *filename, *type;
FILE *ioptr;
The stream named by *ioptr* is closed, if necessary, and then reopened as if by *fopen*. If the attempt to open fails, NULL is returned, otherwise *ioptr*, which will now refer to the new file. Often the reopened stream is *stdin* or *stdout*.

int getc(ioptr)
returns the next character from the stream named by *ioptr*, which is a pointer to a file such as returned by *fopen*, or the name *stdin*. The integer EOF is returned on end-of-file or when an error occurs. The null character is a legal character.

int fgetc(ioptr)
is identical in function to 'getc' but is a genuine function, not a macro.

putc(c, ioptr)
Putc writes the character *c* on the output stream named by *ioptr*, which is a value returned from *fopen* or perhaps *stdout* or *stderr*. The character is returned as value, but EOF is returned on error.

fputc(c, ioptr)
Fputc is identical in function to 'putc' but is a genuine function, not a macro.

fclose(ioptr)
The file corresponding to *ioptr* is closed after any buffers are emptied. A buffer allocated by the IO system is freed. *Fclose* is automatic on normal termination of the program.

fflush(ioptr)
Any buffered information on the (output) stream named by *ioptr* is written out. Output files are normally buffered if and only if they are not directed to the terminal, but *stderr* is unbuffered unless *setbuf* is used.

exit(errcode)
Exit terminates the process and returns its argument as status to the parent. This is a special version of the routine which calls *fflush* for each output file. To terminate without flushing, use *_exit*.

feof(ioptr)
returns non-zero when end-of-file has occurred on the specified input stream.

ferror(ioptr)
returns non-zero when an error has occurred while reading or writing the named stream. The error indication lasts until the file has been closed.

getchar( )
is identical to 'getc(stdin)'.

putchar(c)
is identical to 'putc(c, stdout)'.

*char *gets(s)*
reads characters up to a new-line from the standard input. The new-line character is replaced by a null character. It is the user's responsibility to make sure that the character array *s* is large enough. *Gets* returns its argument, or null if end-of-file or error occurred. Note that this routine is not compatible with *fgets;* it is included for downward compatibility.

*char *fgets(s, n, ioptr)*
reads up to *n* characters from the stream *ioptr* into the character pointer *s*. The read terminates with a new-line character. The new-line character is placed in the buffer followed by a null pointer. The first argument, or a null pointer if error or end-of-file occurred, is returned.

puts(s)
writes the null-terminated string (character array) *s* on the standard output. A new-line is appended. No value is returned. Note that this routine is not compatible with *fputs;* it is included for downward compatibility.

fputs(s, ioptr)
writes the null-terminated string (character array) *s* on the stream *ioptr*. No new-line is appended. No value is returned.

ungetc(c, ioptr)
The argument character *c* is pushed back on the input stream named by *ioptr*. Only one character may be pushed back.

printf(format, a1, . . .)

fprintf(ioptr, format, a1, . . .)

sprintf(s, format, a1, . . .)
Printf writes on the standard output. *Fprintf* writes on the named output stream. *Sprintf* puts characters in the character array (string) named by *s*. The specifications are as usual. There is a new conversion: '%.*n*' converts a double argument in the style of *e* or *f* as most appropriate. The number of significant digits is *n*.

scanf(format, a1, . . .)

fscanf(ioptr, format, a1, . . .)

sscanf(s, format, a1, . . .)
Scanf reads from the standard input. *Fscanf* reads from the named input stream. *Sscanf* reads from the character string supplied as *s*. The specifications are identical to those of the Portable Library.

*fread(ptr, sizeof(*ptr), nitems, ioptr)*

reads *nitems* of data beginning at *ptr* on file *ioptr*. It behaves identically to the Portable Library's *cread*. No advance notification that binary IO is being done is required; when, for portability reasons, it becomes required, it will be done by adding an additional character to the mode-string on the fopen call.

*fwrite(ptr, sizeof(*ptr), nitems, ioptr)*

Like *fread*, but in the other direction.

rewind(ioptr)

rewinds the stream named by *ioptr*. It is not very useful except on input, since a rewound output file is still open only for output.

system(string), atof(s), tmpnam(s), abort(code), "intss( )", "cfree(ptr)", and *"wdleng( )"* are available with specifications identical to those described for the Portable Library.

*char *calloc(n, sizeof(object))*

returns null when no space is available. The space is guaranteed to be 0.

getw(ioptr)

returns the next word from the input stream named by *ioptr*. EOF is returned on end-of-file or error, but since this a perfectly good integer *feof* and *ferror* should be used.

putw(w, ioptr)

writes the integer *w* on the named output stream.

setbuf(ioptr, buf)

Setbuf may be used after a stream has been opened but before IO has started. If *buf* is null, the stream will be unbuffered. Otherwise the buffer supplied will be used. It is a character array of sufficient size: char buf[BUFSIZ];

fileno(ioptr)

returns the integer file descriptor associated with the file.

fseek(ioptr, offset, ptrname)

The location of the next byte in the stream named by *ioptr* is adjusted. *Offset* is a long integer. If *ptrname* is 0, the offset is measured from the beginning of the file; if *ptrname* is 1, the offset is measured from the current read or write pointer; if *ptrname* is 2, the offset is measured from the end of the file. The routine accounts properly for any buffering.

long ftell(iop)

The byte offset, measured from the beginning of the file, associated with the named stream is returned. Any buffering is properly accounted for.

getpw(uid, buf)

The password file is searched for the given integer user ID. If an appropriate line is found, it is copied into the character array *buf*, and 0 is returned. If no line is found corresponding to the user ID then 1 is returned.

strcat(s1, s2)

S1 and *s2* are character pointers. The end (null byte) of the *s1* string is found and *s2* is copied to *s1* starting there. The space pointed to by *s1* must be large enough.

strcmp(s1, s2)

The character strings *s1* and *s2* are compared. The result is positive, zero, or negative according as *s1* is greater than, equal to, or less than *s2* in ASCII collating sequence.

strcpy(s1, s2)

The null-terminated character string *s2* is copied to the location pointed to by *s1*.

strlen(s)

The number of bytes in s up to a null byte is returned. *S* is a character pointer.

The following are macros defined by stdio.h:

isalpha(c)

returns non-zero if the argument is alphabetic.

isupper(c)

returns non-zero if the argument is upper-case alphabetic.

islower(c)

returns non-zero if the argument is lower-case alphabetic.

isdigit(c)

returns non-zero if the argument is a digit.

isspace(c)

returns non-zero if the argument is a spacing character: tab, new-line, carriage return, vertical tab, form feed, space.

toupper(c)

returns the upper-case character corresponding to the lower-case letter c.

tolower(c)

returns the lower-case character corresponding to the upper-case letter c.

FILES

/usr/lib/libS.a
/usr/include/stdio.h