NAME

atan – arc tangent function

SYNOPSIS

jsr r5,atan[**2**]

double atan(x)
double x;

double atan2(x, y)
double x, y;

DESCRIPTION

The *atan* entry returns the arc tangent of fr0 in fr0; from C, the arc tangent of *x* is returned. The range is $-\pi/2$ to $\pi/2$. The *atan2* entry returns the arc tangent of fr0/fr1 in fr0; from C, the arc tangent of *x/y* is returned. The range is $-\pi$ to π.

DIAGNOSTIC

There is no error return.

BUGS

NAME

atof – ascii to floating

SYNOPSIS

double atof(nptr)
char *nptr;

DESCRIPTION

Atof converts a string to a floating number. *Nptr* should point to a string containing the number; the first unrecognized character ends the number.

The only numbers recognized are: an optional minus sign followed by a string of digits optionally containing one decimal point, then followed optionally by the letter **e** followed by a signed integer.

DIAGNOSTICS

There are none; overflow results in a very large number and garbage characters terminate the scan.

BUGS

The routine should accept initial **+**, initial blanks, and **E** for **e**. Overflow should be signalled.

NAME

compar – default comparison routine for qsort

SYNOPSIS

jsr pc,compar

DESCRIPTION

Compar is the default comparison routine called by *qsort* and is separated out so that the user can supply his own comparison.

The routine is called with the width (in bytes) of an element in r3 and it compares byte-by-byte the element pointed to by r0 with the element pointed to by r4.

Return is via the condition codes, which are tested by the instructions ‘‘blt’’ and ‘‘bgt’’. That is, in the absence of overflow, the condition (r0) < (r4) should leave the Z-bit off and N-bit on while (r0) > (r4) should leave Z and N off. Still another way of putting it is that for elements of length 1 the instruction

```
cmpb (r0),(r4)
```

suffices.

Only r0 is changed by the call.

SEE ALSO

qsort (III)

BUGS

It could be recoded to run faster.

NAME

crypt – password encoding

SYNOPSIS

```
mov     $key,r0
jsr     pc,crypt

char *crypt(key)
char *key;
```

DESCRIPTION

On entry, r0 should point to a string of characters terminated by an ASCII NULL. The routine performs an operation on the key which is difficult to invert (i.e. encrypts it) and leaves the resulting eight bytes of ASCII alphanumerics in a global cell called ‘‘word’’.

From C, the *key* argument is a string and the value returned is a pointer to the eight-character encrypted password.

Login uses this result as a password.

SEE ALSO

passwd(I), passwd(V), login(I)

BUGS

NAME

ctime – convert date and time to ASCII

SYNOPSIS

char *ctime(tvec)
int tvec[2];

[from Fortran]
double precision ctime
... = ctime(dummy)

int *localtime(tvec)
int tvec[2];

int *gmtime(tvec)
int tvec[2];

DESCRIPTION

Ctime converts a time in the vector *tvec* such as returned by time (II) into ASCII and returns a pointer to a character string in the form

Sun Sep 16 01:03:52 1973\n\0

All the fields have constant width.

Once the time has been placed into *t* and *t+2*, this routine is callable from assembly language as follows:

```
mov     $t,–(sp)
jsr     pc,_ctime
tst     (sp)+
```

and a pointer to the string is available in r0.

The *localtime* and *gmtime* entries return integer vectors to the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. The value is a pointer to an array whose components are

0 seconds
1 minutes
2 hours
3 day of the month (1-31)
4 month (0-11)
5 year – 1900
6 day of the week (Sunday = 0)
7 day of the year (0-365)
8 Daylight Saving Time flag if non-zero

The external variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, is 5*60*60); the external variable *daylight* is non-zero iff the standard U.S.A. Daylight Saving Time conversion should be applied between the last Sundays in April and October. The external variable *nixonflg* if non-zero supersedes *daylight* and causes daylight time all year round.

A routine named *ctime* is also available from Fortran. Actually it more resembles the *time* (II) system entry in that it returns the number of seconds since the epoch 0000 GMT Jan. 1, 1970 (as a floating-point number).

SEE ALSO

time(II)

BUGS

NAME

ecvt – output conversion

SYNOPSIS

jsr pc,ecvt

jsr pc,fcvt

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
...

DESCRIPTION

Ecvt is called with a floating point number in fr0.

On exit, the number has been converted into a string of ascii digits in a buffer pointed to by r0. The number of digits produced is controlled by a global variable *_ndigits*.

Moreover, the position of the decimal point is contained in r2: r2=0 means the d.p. is at the left hand end of the string of digits; r2>0 means the d.p. is within or to the right of the string.

The sign of the number is indicated by r1 (0 for +; 1 for –).

The low order digit has suffered decimal rounding (i. e. may have been carried into).

From C, the *value* is converted and a pointer to a null-terminated string of *ndigit* digits is returned. The position of the decimal point is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

Fcvt is identical to *ecvt*, except that the correct digit has had decimal rounding for F-style output of the number of digits specified by *_ndigits*.

SEE ALSO

printf(III)

BUGS

NAME

exp – exponential function

SYNOPSIS

jsr r5,exp

double exp(x)
double x;

DESCRIPTION

The exponential of fr0 is returned in fr0. From C, the exponential of *x* is returned.

DIAGNOSTICS

If the result is not representable, the c-bit is set and the largest positive number is returned. From C, no diagnostic is available.

Zero is returned if the result would underflow.

BUGS

NAME

fptrap – floating point interpreter

SYNOPSIS

sys signal; 4; fptrap

DESCRIPTION

Fptrap is a simulator of the 11/45 FP11-B floating point unit. It works by intercepting illegal instruction faults and examining the offending operation codes for possible floating point.

FILES

found in /lib/libu.a; a fake version is in /lib/liba.a

DIAGNOSTICS

A break point trap is given when a real illegal instruction trap occurs.

SEE ALSO

signal(II)

BUGS

Rounding mode is not interpreted. Its slow.

NAME

gerts – Gerts communication over 201

SYNOPSIS

jsr r5,connect
(error return)
...

jsr r5,gerts; fc; oc; ibuf; obuf
(error return)
...

other entry points: **gcset, gout**

DESCRIPTION

The GCOS GERTS interface is so painful that a description here is inappropriate. Anyone needing to use this interface should seek divine guidance.

SEE ALSO

dn(IV), dp(IV), HIS documentation

FILES

found in /lib/libg.a

BUGS

NAME

getarg – get command arguments from Fortran

SYNOPSIS

call getarg (i, iarray, [**, isize**] **)**

... = iargc(dummy)

DESCRIPTION

The *getarg* entry fills in *iarray* (which is considered to be *integer)* with the Hollerith string representing the *i* th argument to the command in which it it is called. If no *isize* argument is specified, at least one blank is placed after the argument, and the last word affected is blank padded. The user should make sure that the array is big enough.

If the *isize* argument is given, the argument will be followed by blanks to fill up *isize* words, but even if the argument is long no more than that many words will be filled in.

The blank-padded array is suitable for use as an argument to setfil (III).

The *iargc* entry returns the number of arguments to the command, counting the first (file-name) argument.

SEE ALSO

exec (II), setfil (III)

BUGS

NAME

getc – buffered input

SYNOPSIS

```
mov     $filename,r0
jsr     r5,fopen; iobuf

fopen(filename, iobuf)
char *filename;
struct buf *iobuf;

jsr     r5,getc; iobuf
(character in r0)

getc(iobuf)
struct buf *iobuf;

jsr     r5,getw; iobuf
(word in r0)
```

[getw not available in C]

DESCRIPTION

These routines provide a buffered input facility. *Iobuf* is the address of a 518(10) byte buffer area whose contents are maintained by these routines. Its format is:

```
ioptr:  .=.+2           / file descriptor
        .=.+2           / characters left in buffer
        .=.+2           / ptr to next character
        .=.+512. / the buffer
```

Or in C,

```
struct buf {
        int fildes;
        int nleft;
        char *nextp;
        char buffer[512];
};
```

Fopen may be called initially to open the file. On return, the error bit (c-bit) is set if the open failed. If *fopen* is never called, *get* will read from the standard input file. From C, the value is negative if the open failed.

Getc returns the next byte from the file in r0. The error bit is set on end of file or a read error. From C, the character is returned; it is –1 on end-of-file or error.

Getw returns the next word in r0. *Getc* and *getw* may be used alternately; there are no odd/even problems. *Getw* is not available from C.

Iobuf must be provided by the user; it must be on a word boundary.

To reuse the same buffer for another file, it is sufficient to close the original file and call *fopen* again.

SEE ALSO

open(II), read(II), putc(III)

DIAGNOSTICS

c-bit set on EOF or error;
from C, negative return indicates error or EOF.

BUGS

NAME

getchar – read character

SYNOPSIS

getchar( )

DESCRIPTION

Getchar provides the simplest means of reading characters from the standard input for C programs. It returns successive characters until end-of-file, when it returns ‘‘\0’’.

Associated with this routine is an external variable called *fin*, which is a structure containing a buffer such as described under *getc* (III).

Normally input via *getchar* is unbuffered, but if the file-descriptor (first) word of *fin* is non-zero, *getchar* calls *getc* with *fin* as argument. This means that

```
fin = open(...)
```

makes *getchar* return (buffered) input from the opened file; also

```
fin = dup(0);
```

causes the standard input to be buffered.

Generally speaking, *getchar* should be used only for the simplest applications; *getc* is better when there are multiple input files.

SEE ALSO

getc (III)

DIAGNOSTICS

Null character returned on EOF or error.

BUGS

–1 should be returned on EOF; null is a legitimate character.

NAME

getpw – get name from UID

SYNOPSIS

getpw(uid, buf)
char *buf;

DESCRIPTION

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

passwd(V)

DIAGNOSTICS

non-zero return on error.

BUGS

It disturbs buffered input via *getchar* (III).

-

NAME

hmul – high-order product

SYNOPSIS

hmul(x, y)

DESCRIPTION

Hmul returns the high-order 16 bits of the product of **x** and **y.** (The binary multiplication operator generates the low-order 16 bits of a product.)

BUGS

NAME

hypot – calculate hypotenuse

SYNOPSIS

jsr r5,hypot

DESCRIPTION

The square root of fr0*fr0 + fr1*fr1 is returned in fr0. The calculation is done in such a way that overflow will not occur unless the answer is not representable in floating point.

DIAGNOSTICS

The c-bit is set if the result cannot be represented.

BUGS

NAME

ierror – catch Fortran errors

SYNOPSIS

if (ierror (*errno* **) .ne. 0) goto** *label*

DESCRIPTION

Ierror provides a way of detecting errors during the running of a Fortran program. Its argument is a run-time error number such as enumerated in *fc* (I).

When *ierror* is called, it returns a 0 value; thus the **goto** statement in the synopsis is not executed. However, the routine stores inside itself the call point and invocation level. If and when the indicated error occurs, a **return** is simulated from *ierror* with a non-zero value; thus the **goto** (or other statement) is executed. It is a ghastly error to call *ierror* from a subroutine which has already returned when the error occurs.

This routine is essentially tailored to catching end-of-file situations. Typically it is called just before the start of the loop which reads the input file, and the **goto** jumps to a graceful termination of the program.

There is a limit of 5 on the number of different error numbers which can be caught.

SEE ALSO

fc (I)

BUGS

There is no way to ignore errors.

NAME

ldiv – long division

SYNOPSIS

ldiv(hidividend, lodividend, divisor)

lrem(hidividend, lodividend, divisor)

DESCRIPTION

The concatenation of the signed 16-bit *hidividend* and the unsigned 16-bit *lodividend* is divided by *divisor*. The 16-bit signed quotient is returned by *ldiv* and the 16-bit signed remainder is returned by *lrem.* Divide check and erroneous results will occur unless the magnitude of the divisor is greater than that of the high-order dividend.

An integer division of an unsigned dividend by a signed divisor may be accomplished by

```
quo = ldiv(0, dividend, divisor);
```

and similarly for the remainder operation.

Often both the quotient and the remainder are wanted. Therefore *ldiv* leaves a remainder in the external cell *ldivr*.

BUGS

No divide check check.

-

NAME

log – natural logarithm

SYNOPSIS

jsr r5,log

double log(x)
double x;

DESCRIPTION

The natural logarithm of fr0 is returned in fr0. From C, the natural logarithm of **x** is returned.

DIAGNOSTICS

The error bit (c-bit) is set if the input argument is less than or equal to zero and the result is a negative number very large in magnitude. From C, there is no error indication.

BUGS

NAME

mesg – write message on typewriter

SYNOPSIS

jsr r5,mesg; <Now is the time\0>; .even

DESCRIPTION

Mesg writes the string immediately following its call onto the standard output file. The string must be terminated by an ASCII NULL byte.

BUGS

NAME

nargs – argument count

SYNOPSIS

nargs( )

DESCRIPTION

Nargs returns the number of actual parameters supplied by the caller of the routine which calls *nargs*.

The argument count is accurate only when none of the actual parameters is *float* or *double*. Such parameters count as four arguments instead of one.

BUGS

As indicated.

NAME

nlist – get entries from name list

SYNOPSIS

```
        jsrr5,nlist; file; list
        ...
file:   <file name\0>; .even
list:   <name1xxx>; type1; value1
        <name2xxx>; type2; value2
        ...
        0

nlist(filename, nl)
char *filename;
struct {
        char    name[8];
        int     type;
        int     value;
} nl[ ];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of a list of 8-character names (null padded) each followed by two words. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are placed in the two words following the name. If the name is not found, the type entry is set to –1.

This subroutine is useful for examining the system name list kept in the file **/usr/sys/unix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(V)

DIAGNOSTICS

All type entries are set to –1 if the file cannot be found or if it is not a valid namelist.

BUGS

NAME

perror – system error messages

SYNOPSIS

perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];

int errno;

DESCRIPTION

Perror produces a short error message describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the newline. *Sys_nerr* is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

Introduction to System Calls

BUGS

-

NAME

pow – floating exponentiation

SYNOPSIS

```
movf    x,fr0
movf    y,fr1
jsr     pc,pow
```

```
double pow(x,y)
double x, y;
```

DESCRIPTION

Pow returns the value of x^y (in fr0). *Pow(0, y)* is 0 for any *y*. *Pow(−x, y)* returns a result only if *y* is an integer.

SEE ALSO

exp(III), log(III)

DIAGNOSTICS

The carry bit is set on return in case of overflow, *pow(0, 0)*, or *pow(−x, y)* for non-integral *y*. From C there is no diagnostic.

BUGS

NAME

printf – formatted print

SYNOPSIS

printf(format, arg$_1$, ...);
char *format;

DESCRIPTION

Printf converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive argument to *printf.*

Each conversion specification is introduced by the character **%**. Following the **%**, there may be

- an optional minus sign '–' which specifies *left adjustment* of the converted argument in the indicated field;
- an optional digit string specifying a *field width;* if the converted argument has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width;
- an optional period ''**.**'' which serves to separate the field width from the next digit string;
- an optional digit string *(precision)* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;
- a character which indicates the type of conversion to be applied.

The conversion characters and their meanings are

d The argument is converted to decimal notation.

o The argument is converted to octal notation. ''0'' will always appear as the first digit.

f The argument is converted to decimal notation in the style ''[–]ddd.ddd'' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed. The argument should be *float* or *double.*

e The argument is converted in the style ''[–]d**.**ddd**e**±dd'' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. The argument should be a *float* or *double* quantity.

c The argument character or character-pair is printed if non-null.

s The argument is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

l The argument is taken to be an unsigned integer which is converted to decimal and printed (the result will be in the range 0 to 65535).

If no recognizable character appears after the **%**, that character is printed; thus **%** may be printed by use of the string **%%**. In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by calling *putchar.*

SEE ALSO

putchar (III)

BUGS

Very wide fields (>128 characters) fail.

NAME

putc – buffered output

SYNOPSIS

```
mov     $filename,r0
jsr     r5,fcreat; iobuf

fcreat(file, iobuf)
char *file;
struct buf *iobuf;

(get byte in r0)
jsr     r5,putc; iobuf

putc(c, iobuf)
int c;
struct buf *iobuf;

(get word in r0)
jsr     r5,putw; iobuf

[putw not available from C]

jsr     r5,flush; iobuf

fflush(iobuf)
struct buf *iobuf;
```

DESCRIPTION

Fcreat creates the given file (mode 666) and sets up the buffer *iobuf* (size 518 bytes); *putc* and *putw* write a byte or word respectively onto the file; *flush* forces the contents of the buffer to be written, but does not close the file. The format of the buffer is:

```
iobuf:  .=.+2           / file descriptor
        .=.+2           / characters unused in buffer
        .=.+2           / ptr to next free character
        .=.+512. / buffer
```

Or in C,

```
struct buf {
        int fildes;
        int nunused;
        char *nxtfree;
        char buff[512];
};
```

Fcreat sets the error bit (c-bit) if the file creation failed (from C, returns −1); none of the other routines returns error information.

Before terminating, a program should call *flush* to force out the last of the output *(fflush* from C).

The user must supply *iobuf,* which should begin on a word boundary.

To write a new file using the same buffer, it suffices to call *[f]flush,* close the file, and call *fcreat* again.

SEE ALSO

creat(II), write(II), getc(III)

DIAGNOSTICS

error bit possible on *fcreat* call.

BUGS

NAME

putchar – write character

SYNOPSIS

putchar(ch)

flush( )

DESCRIPTION

Putchar writes out its argument and returns it unchanged. The low-order byte of the argument is always written; the high-order byte is written only if it is non-null. Unless other arrangements have been made, *putchar* writes in unbuffered fashion on the standard output file.

Associated with this routine is an external variable *fout* which has the structure of a buffer discussed under putc (III). If the file descriptor part of this structure (first word) is not 1, output via *putchar* is buffered. To achieve buffered output one may say, for example,

```
fout = dup(1);          or
fout = fcreat(...);
```

In such a case *flush* must be called before the program terminates in order to flush out the buffered output. *Flush* may be called at any time.

SEE ALSO

putc(III)

BUGS

The *fout* notion is kludgy.

NAME

qsort – quicker sort

SYNOPSIS

(end+1 of data in r2)
(element width in r3)
jsr pc,qsort

qsort(base, nel, width, compar)
char *base;
int (*compar)( );

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The assembly-language version is designed to sort equal length elements. Registers r1 and r2 delimit the region of core containing the array of byte strings to be sorted: r1 points to the start of the first string, r2 to the first location above the last string. Register r3 contains the length of each string. r2–r1 should be a multiple of r3. On return, r0, r1, r2, r3 are destroyed.

The routine compar (q.v.) is called to compare elements and may be replaced by the user.

The C version has somewhat different arguments and the user must supply a comparison routine. The first argument is to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return a negative integer if the first element is to be considered less than the second, a positive integer if the second element is smaller than the first, and 0 if the elements are equal.

SEE ALSO

compar (III)

BUGS

NAME

rand – random number generator

SYNOPSIS

(seed in r0)
jsr pc,srand /to initialize

jsr pc,rand /to get a random number

srand(seed)
int seed;

rand( )

DESCRIPTION

Rand uses a multiplicative congruential random number generator to return successive pseudo-random numbers (in r0) in the range from 1 to $2^{15}-1$.

The generator is reinitialized by calling *srand* with 1 as argument (in r0). It can be set to a random starting point by calling *srand* with whatever you like as argument, for example the low-order word of the time.

WARNING

The author of this routine has been writing random-number generators for many years and has never been known to write one that worked.

BUGS

The low-order bits are not very random.

NAME

reset – execute non-local goto

SYNOPSIS

setexit( )

reset( )

DESCRIPTION

These routines are useful for dealing with errors discovered in a low-level subroutine of a program.

Setexit is typically called just at the start of the main loop of a processing program. It stores certain parameters such as the call point and the stack level.

Reset is typically called after diagnosing an error in some subprocedure called from the main loop. When *reset* is called, it pops the stack appropriately and generates a non-local return from the last call to *setexit.*

It is erroneous, and generally disastrous, to call *reset* unless *setexit* has been called in a routine which is an ancestor of *reset.*

BUGS

NAME

setfil – specify Fortran file name

SYNOPSIS

call setfil (unit**,** hollerith-string **)**

DESCRIPTION

Setfil provides a primitive way to associate an integer I/O *unit* number with a file named by the *hollerith-string.* The end of the file name is indicated by a blank. Subsequent I/O on this unit number will refer to file whose name is specified by the string.

Setfil should be called only before any I/O has been done on the *unit*, or just after doing a **rewind** or **endfile.** It is ineffective for unit numbers 5 and 6.

SEE ALSO

fc (I)

BUGS

There is still no way to receive a file name or other argument from the command line. Also, the exclusion of units 5 and 6 is unwarranted.

NAME

sin – sine, cosine

SYNOPSIS

jsr r5,sin (cos)

double sin(x)
double x;

double cos(x)
double x;

DESCRIPTION

The sine (cosine) of fr0 (resp. **x**), measured in radians, is returned (in fr0).

The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

BUGS

-

NAME

sqrt – square root function

SYNOPSIS

jsr r5,sqrt

double sqrt(x)
double x;

DESCRIPTION

The square root of fr0 (resp. **x**) is returned (in fr0).

DIAGNOSTICS

The c-bit is set on negative arguments and 0 is returned. There is no error return for C programs.

BUGS

No error return from C.

NAME

switch – switch on value

SYNOPSIS

```
        (switch value in r0)
        jsrr5,switch; swtab
        (not-found return)
        ...
swtab:  val1; lab1;
        ...
        valn;labn
        ..;  0
```

DESCRIPTION

Switch compares the value of r0 against each of the val_i; if a match is found, control is transferred to the corresponding lab_i (after popping the stack once). If no match has been found by the time a null lab_i occurs, *switch* returns.

BUGS

NAME

ttyn – return name of current typewriter

SYNOPSIS

jsr pc,ttyn

ttyn(file)

DESCRIPTION

Ttyn hunts up the last character of the name of the typewriter which is the standard input (from *as*) or is specified by the argument *file* descriptor (from C). If *n* is returned, the typewriter name is then ‘‘/dev/tty*n*’’.

x is returned if the indicated file does not correspond to a typewriter.

BUGS

NAME

vt – display (vt01) interface

SYNOPSIS

openvt()

erase()

label(s)
char s[ ];

line(x,y)

circle(x,y,r)

arc(x,y,x0,y0,x1,y1)

dot(x,y,dx,n,pattern)
int pattern[ ];

move(x,y)

DESCRIPTION

C interface routines to perform similarly named functions described in vt(IV). *Openvt* must be used before any of the others to open the storage scope for writing.

FILES

/dev/vt0, found in /lib/libp.a

SEE ALSO

vt (IV)

BUGS