

## CLIB.REL CONTENTS

Module ISGRAP Program 16, Data area 0 Entries: ISGRAP	Test for graphic character class
Module ISDIGI Program 17, Data area 0 Entries: \$ISDIGI, ISDIGI	Test for digit 0123456789
Module ISSPAC Program 14, Data area 0 Entries: ISSPAC Externals: ISSPC\$	Test for space "\t\40\r\n"
Module ISCNTR Program 13, Data area 0 Entries: ISCNTR	Test for control character
Module ISPRIN Program 17, Data area 0 Entries: ISPRIN	Test for printable character
Module ISPUNC Program 41, Data area 0 Entries: ISPUNC	Test for punctuation
Module TOASCII Program 11, Data area 0 Entries: TOASCII	Convert to ASCII range 0-127
Module TOLOWE Program 17, Data area 0 Entries: TOLOWE	Convert to lower case letter
Module TOUPPE Program 17, Data area 0 Entries: TOUPPE	Convert to upper case letter
Module CVUPPE Program 15, Data area 0 Entries: \$CVUPP, CVUPPE Externals: NAM.U	Convert whole string to upper case
Module ABS Program 11, Data area 0 Entries: ABS Externals: C.NEG	Integer absolute value
Module MAX Program 16, Data area 0 Entries: MAX Externals: C.LT	Integer maximum value

## CLIB.REL CONTENTS

Module CCMAIN	Main stub required for each main program
Program 132, Data area 22	
Entries: \$LM, A EXIT, CBUF, CC_CCP, CMODE, CTLB, CTLB., CTLB_EXIT, FIN, FOUT, P FIN, P FOUT	
Externals: \$END, CEXIT_, COPEN_, C_MCLN, EX_SPC, MAIN, TOKENS, TOT_SZ	
Module ERRNO	Error number variable
Program 132, Data area 2	
Entries: ERRNO	
Module CMCLN	C command line copier.
Program 67, Data area 12	
Entries: C_MCLN	
Externals: CBUF	
Module CORE	Gets memory from sbrk().
Program 39, Data area 0	
Entries: CORE	
Externals: EXIT, SBRK	
Module FINFOU	Stdin and stdout returns.
Program 8, Data area 0	
Entries: FIN , FOUT	
Externals: FIN, FOUT	
Module ISASCII	Test for ASCII character class
Program 13, Data area 0	
Entries: ISASCII	
Module ISALNU	Test for alphanumeric character class
Program 13, Data area 0	
Entries: ISALNU	
Externals: \$ISALP, \$ISDIG	
Module ISALPH	Test for alphabetic character class
Program 13, Data area 0	
Entries: \$ISALP, ISALPH	
Externals: \$ISLOW, \$ISUPP	
Module ISUPPE	Test for upper case letter
Program 17, Data area 0	
Entries: \$ISUPP, ISUPPE	
Module ISLOWE	Test for lower case letter
Program 17, Data area 0	
Entries: \$ISLOW, ISLOWE	
Module ISXDIG	Test hex digit 0123456789ABCDEFabcdef
Program 18, Data area 0	
Entries: ISXDIG	
Externals: \$ISDIG	

## CLIB.REL CONTENTS

Module GETBYT Program 15, Data area 0 Entries: GETBYT Externals: CMODE, GETCHA	Special get byte no echo
Module STRCMP Program 28, Data area 0 Entries: STRCMP Externals: RN\$, S.1	K&R string compare
Module STRCPY Program 21, Data area 0 Entries: STRCPY	K&R string copy
Module STRCAT Program 28, Data area 0 Entries: STRCAT	K&R string concatenate
Module STRNCM Program 42, Data area 0 Entries: STRNCM Externals: RN\$, S.1	K&R string n-compare
Module STRNCP Program 36, Data area 2 Entries: STRNCP	K&R string n-copy with null fill. WARNING: May not terminate string.
Module STRNCA Program 45, Data area 2 Entries: STRNCA	K&R string n-concatenate
Module STRCSP Program 35, Data area 0 Entries: STRCSP	Harbison & Steele strcspn()
Module STRPBR Program 32, Data area 0 Entries: STRPBR	Harbison & Steele strpbrk()
Module STRPOS Program 21, Data area 0 Entries: STRPOS Externals: \$RM	Harbison & Steele strpos()
Module STRRPB Program 42, Data area 0 Entries: STRRPB	Harbison & Steele strrpbrk()
Module STRRPO Program 31, Data area 0 Entries: STRRPO Externals: \$RM	Harbison & Steele strrpos()

## CLIB.REL CONTENTS

Module MIN	Integer minimum value
Program 16, Data area 0	
Entries: MIN	
Externals: CLT	
Module ATOI	ASCII text to integer 16-bit
Program 31, Data area 0	
Entries: ATOI	
Externals: \$CVI, C.NEG, SOB	
Module TOINT	Hex digit to integer byte
Program 34, Data area 0	
Entries: TOINT	
Module ITOA	Integer 16 bit to ASCII string
Program 34, Data area 0	
Entries: ITOA	
Externals: \$NUMST	
Module REVERS	Reverse string in place
Program 34, Data area 0	
Entries: REVERS	
Externals: HLCPDE	
Module GETS	Get string from console
Program 17, Data area 0	
Entries: GETS	
Externals: GETLN	
Module GETLN	Get string from console with max limit
Program 28, Data area 2	
Entries: GETLN	
Externals: GETLIN	
Module GETLIN	Get line from console K&R.
Program 49, Data area 0	
Entries: GETLIN	
Externals: GETCHA, HLCPDE, RH\$	
Module FPUTS	Standard K&R fputs() to stream
Program 33, Data area 0	
Entries: FPUTS	
Externals: \$RM, PUTC, RN\$	
Module FGETS	Standard K&R fgets().
Program 59, Data area 0	
Entries: FGETS	
Externals: GETC, HLCPDE, RN\$	

## CLIB.REL CONTENTS

Module CONOUT	Console only character output with multiple arguments.
Program 30, Data area 0	
Entries: CONO 1, CONO 2	
Externals: \$CON, PROCL.	
Module TYPE	Re-direction string output.
Program 73, Data area 0	Multiple arguments.
Entries: TYPE 1, TYPE 2, TYPE 3	
Externals: PROCL, PUTCHA	
Module PRINTF	Printf() header, K&R hispeed with many features omitted. See also CLIBM printf.
Program 31, Data area 0	
Entries: PRTF 1, PRTF 2	
Externals: PRNTF., PUTCHA	
Module FPRINT	K&R fprintf(), see printf() above.
Program 42, Data area 0	
Entries: FPRT 1, FPRT 2	
Externals: PRNTF., PUTC	
Module SPRINT	K&R sprintf(), see printf() above.
Program 47, Data area 0	
Entries: SPRT 1, SPRT 2	
Externals: PRNTF.	
Module PRNTF	Main printf routine.
Program 250, Data area 25	
Entries: FMTLOC, PRNTF, PRNTF.	
Externals: \$CVI, \$NUMST, \$SETPA, H. HLCPDE, PROCL.	
Module SETPAD	Pad character set routine for printf().
Program 79, Data area 0	
Entries: \$SETPA	
Externals: HLCPDE, S.1	
Module NUMSTR	Number conversion for printf().
Program 169, Data area 0	
Entries: \$NUMST	
Externals: C.NEG	
Module CVI	Integer conversion for printf().
Program 29, Data area 0	
Entries: \$CVI	
Module PROCL	Process list of multiple arguments.
Program 29, Data area 0	
Entries: PROCL.	
Externals: HLCPDE	
Module RESET	Reset disk system
Program 6, Data area 0	
Entries: RESET	

## CLIB.REL CONTENTS

Module STRSPN Program 39, Data area 0 Entries: STRSPN	Harbison & Steele strspn()
Module INDEX Program 17, Data area 0 Entries: INDEX, STRCHR Externals: RN\$	Unix-style index()
Module RINDEX Program 31, Data area 0 Entries: RINDEX Externals: RN\$	Unix-style rindex()
Module INSTR Program 67, Data area 0 Entries: INSTR Externals: COMP\$, RN\$	Microsoft MBASIC-style instr()
Module MIDSTR Program 51, Data area 0 Entries: MIDSTR Externals: RN\$	Faster instr() for 2 arguments.
Module COMP Program 39, Data area 0 Entries: COMP COMP\$ Externals: \$RM, RN\$	Server for instr() function above
Module STRLEN Program 16, Data area 0 Entries: STRLEN	K&R string length
Module FILLCH Program 23, Data area 0 Entries: FILLCH	PASCAL-style fill memory. Arguments like strncpy().
Module MOVMEM Program: 10, Data area 0 Entries: MOVMEM Externals: MOVEME	Memory move with reversed arg order.
Module MOVEME Program 53, Data area 0 Entries: MOVEME Externals: HLCPDE, MOVE\$	Memory move without overlap faults, uses Z80 block move on Z80 CPU. Arguments are like strncpy().
Module ERROR Program 50, Data area 0 Entries: ERROR, ERR 1, ERR 2 Externals: \$CON, PROCL.	Console-only string out. Mult-args.

# CLIB.REL CONTENTS

Module PUTC	Put character ASCII mode, K&R
Program 135, Data area 0	
Entries: PUTC	
Externals: \$PCH, \$RM, FFB, FFLUSH, IOBIN, IOBUF, IOIND, IORW, IOSIZE,	
MAXCHN, PC_BIN	
Module PCH	Subroutine for putc()
Program 40, Data area 0	
Entries: \$PCH	
Externals: \$PU, IOPWRI, PC_BIN	
Module OR	Logical OR function
Program 10, Data area 0	
Entries: 0.	
Module XOR	Exclusive OR function
Program 10, Data area 0	
Entries: X.	
Module AND	Logical AND function
Program 10, Data area 0	
Entries: A.	
Module HLDE	Test HL=DE
Program 14, Data area 0	
Entries: N., N.1	
Module CMPSIG	Compare signs, register level
Program 45, Data area 0	
Entries: C.GE, C.GT, C.LE, C.LT	
Module ASR	Arithmetic shift right
Program 14, Data area 0	
Entries: C.ASR	
Module USR	Unsigned shift right
Program 13, Data area 0	
Entries: C.USR	
Module ASL	Arithmetic shift left
Program 7, Data area 0	
Entries: C.ASL	
Module Q	Store HL at address on stack
Program 8, Data area 0	
Entries: Q.	
Module CMULT	Multiplication, signed
Program 28, Data area 0	
Entries: C.MULT	

## CLIB.REL CONTENTS

Module PUTCHA	K&R putchar()
Program 26, Data area 0	
Entries: PUTCHA	
Externals: \$PCH, FOUT, PUTC	
Module GETCHA	K&R getchar()
Program 14, Data area 0	
Entries: GETCHA	
Externals: \$B8, FIN, GETC	
Module UNGETC	K&R stream unget character
Program 119, Data area 0	
Entries: UNGETC	
Externals: \$UNGET, FIN, IOCH, IOFCB, IOIND, IOMODE, IOSECT, IOSIZE, MAXCHN	
Module GETCBI	Get character binary
Program 49, Data area 0	
Entries: GETCBI	
Externals: CMODE, GC_BIN, GETC	
Module GETC	Get character ASCII mode
Program 165, Data area 1	
Entries: GC_BIN, GETC	
Externals: \$GCH, \$RM, FFB, FFLUSH, IOBIN, IOBUF, IOIND, IONCHX, IOSIZE, MAXCHN, READ	
Module KEYSTA	Keyboard and console buffer status
Program 19, Data area 0	
Entries: KEYSTA	
Externals: \$KEYST, \$RM, CBUF, CCNT	
Module GETPUT	Subroutine for device input
Program 120, Data area 2	
Entries: \$B8, \$GCH, \$RLIN, \$UNGET, CCNT, CCTLCK	
Externals: \$BREAK, \$CONIN, \$PU, \$RADD, \$RM, CBUF, CMODE, CTLB., CTLCK, IOPREA,	
Module SELDSK	Select disk drive
Program 13, Data area 0	
Entries: SELDSK	
Module PUTCBI	Put character binary
Program 26, Data area 0	
Entries: PUTCBI	
Externals: PC_BIN PUTC	

## CLIB.REL CONTENTS

Module CDIV	Division, signed
Program 72, Data area 0	
Entries: C.DIV, C.UDV	
Externals: C.NEG	
Module SWITCH	Switch case subroutine
Program 26, Data area 0	
Entries: .SWITC	
Module CNEG	Compute HL = -HL
Program 8, Data area 0	
Entries: C.COM, C.NEG	
Module CMOT	Return TRUE if HL==0
Program 10, Data area 0	
Entries: C.NOT	
Externals: E.2	
Module UGE	Unsigned >= test
Program 18, Data area 0	
Entries: C.UGE, C.ULE	
Module UGT	Unsigned > test
Program 14, Data area 0	
Entries: C.UGT, C.ULT	
Module MEMSIZ	Memory size for heap space managed by sbrk().
Program 25, Data area 0	
Entries: HIGHME, LOWMEM, MEMSIZ	
Externals: \$LM, \$RM, HLCODE, S.1	
Module CFS	Compute file size, virtual
Program 39, Data area 0	
Entries: CFS	
Externals: \$RM, .XFCB, H.	
Module RENAME	Rename a file to new name
Program 38, Data area 0	
Entries: RENAME	
Externals: .XFCB	
Module UNLINK	Erase a disk file by name
Program 26, Data area 0	
Entries: UNLINK	
Externals: .XFCB	
Module LOGGED	Report currently logged disk
Program 9, Data area 0	
Entries: LOGGED	
Module LOGINV	Return 16 bit vector of logged drives
Program 5, Data area 0	
Entries: LOGINV	

**Module FINDDI**  
Program 58, Data area 0  
Entries: FINDFI, FINDNE  
Externals: \$RM, .XFCB

**Module FIXFIL** Fix a file name from template string  
Program 102, Data area 0  
Entries: FIXFIL  
Externals: .XFCB, DECODEF, MOVE\$

**Module DECODEF** Decode a file control block to a file name.  
Program 70, Data area 0  
Entries: DECODEF

**Module CHMODE** Change to/from character mode  
Program 9, Data area 0  
Entries: CHMODE  
Externals: CMODE

**Module MAKEFC** Make a file control block  
Program 12, Data area 0  
Entries: MAKEFC  
Externals: .XFCB

**Module TOKENS** Token decoder for C command line  
Program 132, Data area 0  
Entries: ISQUO\$, TOKEN\$, TOKENS  
Externals: ISSPC\$, SOB\$

**Module COPEN** Re-direction file fin/fout open routine  
Program 51, Data area 0  
Entries: COPEN  
Externals: A\_EXIT, FOPEN

**Module CEXIT** Re-direction file fclose() routine  
Program 12, Data area 0  
Entries: CEXIT  
Externals: FCLOSE, FOUT

**Module FOPEN** Standard C/80 fopen(), not K&R, but close.  
Program 219, Data area 5  
Entries: FOPEN, FOPEN  
Externals: \$PU, .CPCI, .XFCB, IOBIN, IOBUF, IOCH, IODEV, IOEND, IOFCB, IOMODE, IONCHX, IORW, IOSECT, IOSIZE, MAXCHN, RN\$, SBRK

**Module FCLOSE** Standard K&R fclose() for streams.  
Program 73, Data area 0  
Entries: FCLOSE  
Externals: \$PU, FFLUSH, IOCH, IOFCB, IOPEOF, IOPWRI, MAXCHN, RN\$

## CLIB.REL CONTENTS

Module REWIND                               Rewind open stream  
Program 42, Data area 0  
Entries: REWIND  
Externals: FFLUSH, IOFCB, IOIND, IOSECT

Module FFLUSH                               Non-standard file flush - does not put  
Program 133, Data area 0                   FCB into directory. See CLIBU.REL.  
Entries: FFLUSH  
Externals: \$RM, HLCPDE, IOBIN, IOBUF, IOFCB, IOIND, IOMODE, IORW,  
IOSECT, RN\$, WRITE

Module FFB                                   File file buffer from disk  
Program 96, Data area 0  
Entries: FFB  
Externals: IOBIN, IOBUF, IOIND, IONCHX, IORW, IOSIZE, READ

Module WRITE                               C/80 write binary. See writea, writeb.  
Program 186, Data area 30  
Entries: READ, READ , WRITE, WRITE  
Externals: CMODE, CTLCK, IOFCB, IOSECT, MAXCHN, RN\$, S.1

Module CON                                  Console output, uses BDOS.  
Program 9, Data area 0  
Entries: CON

Module CONBUF                              Subroutines for console buffer use.  
Program 59, Data area 0                   See putc(), getc().  
Entries: \$BREAK, \$RADD, CTLCK  
Externals: A\_EXIT, CBUF, CMODE, CTLB., \$CON, \$CONIN, \$KEYST

Module CONIO                               Console input & output routines  
Program 40, Data area 0  
Entries: \$CON, \$CONIN, \$KEYST  
Externals: CMODE

Module XFCB                                Make file control block, low level  
Program 112, Data area 0  
Entries:  
  .XFCB  
Externals: ISSPC\$, NAM.U

Module CPC1                                Subroutine for fopen  
Program 36, Data area 0  
Entries:  
  .CPC1  
Externals: IOFCB, RN\$, SBRK

Module SFREE                               Frees space set up by sbrk().  
Program 108, Data area 0  
Entries: SFREE  
Externals: \$END, \$LM, H., HLCPDE, IOBIN, IOBUF, IOFCB, IOSIZE, MAXCHN, RN\$, S.1

## CLIB.REL CONTENTS

Module SBRK	Gives heap address for space request or -1 on failure
Program 38, Data area 0	
Entries: SBRK	
Externals: \$LM, \$RM, HLCPDE, S.1	
Module BRK	Sets program break address, see Banahan and Rutter and CP/M-68k manual.
Program 23, Data area 0	
Entries: BRK	
Externals: \$LM, \$RM, HLCPDE	
Module \$PU	Physical unit subroutine
Program 22, Data area 0	
Entries: \$PU	
Externals: \$RM	
Module IOTABL	Data for all file I/O and buffers.
Program 22, Data area 161	
Entries: IOBIN, IOBUF, IOCH, IODEV, IOEND, IOFCB, IOIND, IOMODE, IONCHX, IOPCHA, IOPEOF, IOPREA, IOPWRI, IORW, IOSECT, IOSIZE, IOTMP, MAXCHN	
Externals: \$END, \$OFF1, \$OFF2, \$OFF3, \$OFF4, \$OFF5, \$OFF6, \$SIZ1, \$SIZ2, \$SIZ3, \$SIZ4, \$SIZ5, \$SIZ6, \$TAG1, \$TAG2, \$TAG3, \$TAG4, \$TAG5, \$TAG6	
Module NAMU	Upper case name converter for files
Program 11, Data area 0	
Entries: NAM.U	
Module E	Test HL=0 in various ways
Program 25, Data area 0	
Entries: E., E.0, E.1, E.2, E.F, E.T	
Module H	Fetch to HL from address in HL. Register level function.
Program 5, Data area 0	
Entries: H.	
Module HLCPDE	Compare register HL to DE
Program 6, Data area 0	
Entries: HLCPDE	
Module SOB	Skip over blanks in a string
Program 13, Data area 0	
Entries: SOB, SOB\$	
Externals: ISSPC\$	
Module FNB	Find next blank in a string
Program 15, Data area 0	
Entries: FNB, FNBS\$	
Externals: ISSPC\$	
Module ISSPC	Test for space characters, low level
Program 12, Data area 0	
Entries: ISSPC\$	

## CLIB.REL CONTENTS

Module PEEKB Program 8, Data area 0 Entries: PEEKB	Peek at address for byte return
Module PEEKW Program 9, Data area 0 Entries: PEEKW	Peek address for 16 bit return
Module POKEB Program 8, Data area 0 Entries: POKEB	Poke byte at address
Module POKEW Program 11, Data area 0 Entries: POKEW	Poke 16 bit word at address
Module BDOS Program 8, Data area 0 Entries: BDOS Externals: CCB DOS	Standard BDOS interface
Module BIOS Program 12, Data area 0 Entries: BIOS Externals: CCB BIOS	Standard BIOS interface
Module CCB DOS Program 40, Data area 0 Entries: CCB DOS	Special reverse argument BDOS interface with all registers.
Module CCB BIOS Program 52, Data area 0 Entries: CCB BIOS	Special reverse argument BIOS interface with all registers and special returns, depending on function number.
Module SETJMP Program 38, Data area 0 Entries: LONGJM, SETJMP	Set jump companion for long jump return
Module MOVE Program 25, Data area 0 Entries: MOVE\$	Low level memory move
Module SUBT16 Program 10, Data area 0 Entries: S., S.1	16 bit subtract, low level
Module RETURN Program 8 Data area 0 Entries: \$RM, RN\$	Zero and -1 standard returns

## CLIB.REL CONTENTS

<b>Module G</b> Program 6, Data area 0 Entries: C.SXT, G.	Fetch character and sign extension
<b>Module IMPORT</b> Program 14, Data area 0 Entries: IMPORT	Input byte from CPU port
<b>Module OUTPOR</b> Program 17, Data area 0 Entries: OUTPOR	Output byte to CPU port
<b>Module CTST</b> Program 6, Data area 0 Entries: C.TST	Utility sign test, low level
<b>Module BINFLA</b> Program 6, Data area 2 Entries: GC_BIN, PC_BIN	Binary flags for I/O
<b>Module END</b> Program 6, Data area 0 Entries: \$END	End routine, creates address for sbrk() to manage heap space.

## CLIB X.REL CONTENTS

Module SSORT2	Shell sort for pointer tables
Program 215, Data area 16	
Entries: SSORT2	
Externals: \$RM	
Module SSORT3	Shell sort for MBASIC structures
Program 237, Data area 0	
Entries: SSORT3	
Externals: \$RM	
Module NUMSOR	Distribution sort (mathsort)
Program 184, Data area 10	
Entries: NUMSOR	
Module TIMER	H89/Z100 timer utility 1/10 second
Program 209, Data area 2	
Entries: TIMER	
Externals: C.UDV, H., ITOA, PEEKW, PUTCHA, Q., S.	
Module DSORT1	Distribution sort, 16-bit numbers
Program 176, Data area 10	
Entries: DSORT1	
Externals: MOVE\$	
Module DSORT	Distribution sort, single byte
Program 152, Data area 10	
Entries: DSORT	
Externals: MOVE\$	
Module RAWIO	rdDISK, wrDISK direct-disk
Program 164, Data area 0	
Entries: RDDISK, WRDISK	
Module WAITZ	H89/Z100 timeout wait routine
Program 25, Data area 0	
Entries: WAITZ	
Module BINARY	Binary search for integers
Program 96, Data area 8	
Entries: BINARY	
Externals: \$RM, C.LE, C.LT, H., S.1	
Module SBINAR	Binary search for strings
Program 99, Data area 8	
Entries: SBINAR	
Externals: \$RM, C.LE, C.LT, H., STRCMP	
Module SEEKZ	Low-level integer seek,fte1l,fte1lr
Program 1001, Data area 24	
Entries: FTELL, FTELLR, SEEK	
Externals: .SWITC, C.DIV, C.GT, C.LT, C.MULT, C.NOT, C.ULT, CCBDO\$ E., E.0, FFB, FFLUSH, G., H., IOBIN, IOEND, IOFCB, IOIND, IOMODE, IOSECT, MOVEME, Q., S.	

## CLIBX.REL CONTENTS

Module SFATN1 Scanf module  
Program 245, Data area 10  
Entries: SF ATN, SF OP1  
Externals: .SWITC, ATOF, C.MULT, H., HI.BL, I..L, L.ADD, L.MUL, LLONG., Q., SLONG., ST4.

Module SFATN2 Scanf module  
Program 217, Data area 10  
Entries: SF ATN, SF OP2  
Externals: .SWITC, C.MULT, H., HI.BL, I..L, L.ADD, L.MUL, LLONG., Q., SLONG., ST4.

Module SFATN3 Scanf module  
Program 125, Data area 6  
Entries: SF ATN, SF OP3  
Externals: .SWITC, ATOF, C.MULT, H., Q., SLONG.

Module SCANF Main scanf routine  
Program 1695, Data area 40  
Entries: F SCAN, SCAN F, STK PO, S SCAN  
Externals: .SWITC, C.GT, C.MULT, C.NOT, C.TST, C.UGE, E., E.O, G., GETC, GETCHA, H., INDEX, ISDIGI, ISSPAC, ISXDIG, N., SF ATN, TOLOWE, UNGETC

Module SFATN4 Scanf module  
Program 97, Data area 6  
Entries: SF ATN, SF OP4  
Externals: .SWITC, C.MULT, H., Q.

Module RUN Run a CP/M command line.  
Program 51, Data 0  
Entries: RUN  
Externals: .XFCB, CBUF, CHAIN, FNBS, INSERT, SOB\$, STRCPY

Module CHAIN Chain from FCB to tpa.  
Program 94, Data 0  
Entries: CHAIN  
Externals: MOVE\$

Module INSERT Insert command tail into default  
Program 58, Data 0 CP/M buffers.  
Entries: INSERT  
Externals: .XFCB, CVUPPE, FNBS, MOVE\$, SOB\$, STRLEN

Module GETDDI Get directory info strings.  
Program 387, Data 8  
Entries: GETDDI  
Externals: C.GT, C.LE, C.LT, C.NOT, DECODEF, E., E.O, FINDFI, FINDNE, H., LOGGED, N., STRCMP, STRCPY

Module GETDFR Get disk free space info.  
Program 400, Data 20  
Entries: GETDFR  
Externals: A., C.ASL, C.DIV, C.GE, C.LE, C.MULT, C.NOT, C.UDV, CCBDS, CCBIOS, E.O, H., Q., TOUPPE

# CLIBM.REL CONTENTS

Module PEEKL	Peek at address, long return
Program 8, Data area 0	
Entries: PEEKL	
Externals: LLONG.	
Module POKEL	Poke long at address
Program 15, Data area 0	
Entries: POKEL	
Externals: LLONG., SLONG.	
Module PFOP1	Printf module
Program 5, Data area 0	
Entries: CVLTOA, PF_OP1	
Module PFOP2	Printf module
Program 5, Data area 0	
Entries: CVFTOA, PF_OP2	
Module PF	Printf main routine, long & float.
Program 1202, Data area 42	
Entries: PRNT 1, PRNT 2, PRNT 3, PRNT 4	
Externals: .SWITC, A., C.GE, C.GT, C.MULT, C.NEG, C.NOT, C.UDV, C.UGE, C.ULT, C.USR, CVFTOA, CVLTOA, E.O, H., MIN, PUTC, PUTCHA, S., STRLEN	
Module CVLTOA	Convert long to ASCII for printf()
Program 554, Data area 22	
Entries: CVLTOA	
Externals: BL.HU, C.NOT, C.ULT, CL.LT, E.O, H., HI.BL, HU.BL, L.AND, L.ASR, L.DIV, L.MUL, L.NEG, L.SUB, LLONG., LONG.O, SLONG., SWAP4.	
Module CVFTOA	Convert float to ASCII, printf()
Program 195, Data area 12	
Entries: CVFTOA	
Externals: C.LT, FTOA, G., LLONG., SLONG., STRLEN	
Module ATOL	ASCII to long conversion, base 10
Program 124, Data area 8	
Entries: ATOL	
Externals: C..L, I..L, ISDIGI, L.ADD, L.MUL, LLONG., SLONG., SOB, ST4.	
Module LTOA	Long to ASCII, base 10. Fast version.
Program 337, Data area 16	
Entries: LTOA	
Externals: C.GT, C.TST, CL.GE, CL.GT, CL.LE, CL.LT, E.O, H., HI.BL, ITOA, L.NEG, L.SUB, LLONG., SLONG., SWAP4.	
Module LABS	Long absolute value
Program 25, Data area 0	
Entries: LABS	
Externals: CL.GT, HI.BL, L.NEG	

# CLIBM.REL CONTENTS

Module FABS	Float absolute value
Program 25, Data area 0	
Entries: FABS	
Externals: CF.GT, F.NEG, HI.BF	
Module FTOA	Float to ASCII, 4 arguments
Program 470, Data area 2	
Entries: FTOA, FTOA	
Externals: DIV10., FADD., FADDA., FCMP., FLNEG., INXHR., LLONG., MOVFR., MOVRF., MOVRM., MUL10., QINT., SIGN.	
Module ATOF	ASCII to float, Unix-style
Program 199, Data area 0	
Entries: ATOF	
Externals: DIV10., FADD., FLNEG., FLOAT., MOVRF., MUL10., PUSHF., ZERO.	
Module FSTACK	FLIBRARY from Software Toolworks
Program 73, Data area 0	
Entries: C..F, C@F, F..C, F..I, F..L, F..U, F@C, F@I, F@L, F@U, F STAK, I..F, I@F, L..F, L@F, U..F, U@F	
Externals: BF.BL, BF.HC, BF.HI, BF.HU, BL.BF, C.1632, C.3216, G., HC.BF, HER.32, HI.BF, HU.BF, MOVRM., SLONG.	
Module FLTLIB	FLIBRARY from Software Toolworks
Program 1112, Data area 15	
Entries: BF.BL, BF.HC, BF.HI, BF.HU, BF@BL, BF@HC, BF@HI, BF@HU, BL.BF, BL@BF, CF.EQ, CF.GE, CF.GT, CF.LE, CF.LT, CF.NE, CF@EQ, CF@GE, CF@GT, CF@LE, CF@LT, CF@NE, DIV10., DUM, ERRCOD, F.ADD, F.DIV, F.MUL, F.NEG, F.NOT, F.SUB, FDADD, FDIV, FONU, FONEG, FONOT, FOSUB, FACL , FACL 1, FACL 2, FAC , FAC 1, FADD., FADDA., FCMP., FCMP@, FDIV A, FDIV B, FDIV C, FDIV G, FLNEG., FLOAT., FLT.0, FLT@0, FLT PK, FMET 1, FMET 2, HC.BF, HC@BF, HI.BF, HI@BF, HU.BF, HU@BF, INXHR., MOVFM., MOVFM@, MOVFR., MOVFR@, MOVMF., MOVMF@, MOVRF., MOVRF@, MUL10., PUSHF., QINT., SAVE , SAVE 1, SIGN., ZERO.	
Externals: C.NEG, C.SXT, EQ.4, G., HC.BL, HI.BL, HU.BL, L.ADD, L.NEG, LLONG., MOVRM., NEQ.4, SLONG.	
Module LANDSH	FLIBRARY from Software Toolworks
Program 141, Data area 0	
Entries: L.AND, L.ASL, L.ASR, L@AND, L@ASL, L@ASR, L_SHIF	
Externals: G.	
Module LCOMP	FLIBRARY from Software Toolworks
FLIBRARY from Software Toolworks	
Program 84, Data area 0	
Entries: CL.GE, CL.GT, CL.LE, CL.LT, CL@GE, CL@GT, CL@LE, CL@LT, L_COMP	
Externals: G., LLONG., LONG.0, SLONG., SWAPS.	
Module LMISC	FLIBRARY from Software Toolworks
Program 372, Data area 0	
Entries: L.DIV, L.MOD, L.MUL, L.OR, L.XOR, L@DIV, L@MOD, L@MUL, L@OR, L@XOR, L_MDIV	
Externals: G.	

## **C L I B M . R E L C O N T I N U E D . . .**

**Module LADSUB** FLIBRARY from Software Toolworks  
Program 46, Data area 0  
Entries: L.ADD, L.COM, L.NEG, L.SUB, L@ADD, L@COM, L@NEG, L@SUB, L.\_ADSB  
Externals: G.

**Module LSTACK** FLIBRARY from Software Toolworks  
Program 104, Data area 0  
Entries: C..L, C.1632, C.3216, C@CL, HER.32, I..L I@CL, L..C, L..I,  
L..U, L@C, L@I, L@U, L STAK, U..L, U@CL  
Externals: BL.HC, BL.HI, BL.HU, G., HC.BL, HI.BL, MOVRM., SLONG.

**Module FOURB** FLIBRARY from Software Toolworks  
Program 107, Data area 0  
Entries: EQ.4, EQ@4, FOUR B, L.NOT, L@NOT, LLONG., LLONG@, LONG.0,  
LONG@0, MOVRM., MOVR@0, NEQ.4, NEQ@4, SLONG., SLONG@, ST4., ST4@,  
SWAP4., SWAP4@, SWAPS., SWAPS@  
Externals: G.

**Module HTOBL** FLIBRARY from Software Toolworks  
Program 35, Data area 0  
Entries: BL.HC, BL.HI, BL.HU, BL@HC, BL@HI, BL@HU, HC.BL, HC@BL, HI.BL,  
HI@BL, HU.BL, HU@BL, I LONG  
Externals: C.SXT G.

## **C L I B U . R E L - U N I X L I B R A R Y**

**Module UFLUSH** Unix-style fflush  
Program 54, Data area 0  
Entries: UFLUSH  
Externals: \$RM, CCBDOS, FFLUSH, H., IOCH, IOFCB, MAXCHN, RN\$

**Module ABORT** Abort debugger service  
Program 6, Data area 0  
Entries: ABORT

**Module ACCESS** Access system call  
Program 105, Data area 4  
Entries: ACCESS  
Externals: C.NOT E.0 GETATT H.

**Module CALLOC** Memory allocation  
Program 34, Data area 0  
Entries: CALLOC  
Externals: C.MULT, FILLC, MALLOC

**Module CHMOD** Change file mode  
Program 87, Data area 2  
Entries: CHMOD  
Externals: H., SETATT

## CLIBU.REL CONTENTS

Module CHOWN	Change file owner
Program 26, Data area 0	
Entries: CHOWN	
Externals: FINDFI, H.	
Module CLOSE	Close file, low-level file descriptor
Program 3, Data area 0	
Entries: CLOSE	
Externals: FCLOSE	
Module CREAT	Create file, low-level file descriptor
Program 122, Data area 2	
Entries: CREATA, CREATB	
Externals: C.TST, FOPEN, H., IOBIN, IOMODE	
Module CTYPE	Character class functions, array type
Program 147, Data area 0	
Entries: CTYPE_, ISTYPE	
Module EXPAND	Command line expansion
Program 495, Data area 12	
Entries: EXPAND	
Externals: C.GT, CORE, CVUPPE, DECODEF, E.O, FINDFI, FINDNE, H., INDEX, N., Q., STRCMP, STRCPY, STRLEN	
Module ETEXT	Etext, End, Edata variables
Program 6, Data area 0	
Entries: EDATA, END, ETEXT	
Externals: \$END, TOT_SZ	
Module OPENB	Open low-level by file descriptor, binary mode.
Program 106, Data area 2	
Entries: OPENB	
Externals: C.GE, C.LE, E.O, FOPEN, H.	
Module OPENA	Open low-level by file descriptor, ASCII
Program 103, Data area 2	
Entries: OPENA	
Externals: C.GE, C.LE, E.O, FOPEN, H.	
Module FDOPEN	File descriptor open
Program 83, Data area 0	
Entries: FDOPEN	
Externals: C.GT, C.LE, DECODEF, E.O, FREOPA, H., IOFCB, MAXCHN	
Module FREOPB	Stream re-open binary mode
Program 179, Data area 6	
Entries: FREOPB	
Externals: E.O, FREOPA, H.	

## CLIBU.REL CONTENTS

Module FREOPA	Stream re-open ASCII mode
Program 289, Data area 4	
Entries: FREOPA	
Externals: C.GT, C.LE, E., E.O, FCLOSE, FIN, FOPENA, FOUT, H., IOCH, MAXCHN	
Module FEOF	End of file check
Program 90, Data area 2	
Entries: FEOF	
Externals: C.NOT, E.O, GETCBI, H., IOMODE, UNGETC	
Module FERROR	File error reporting
Program 4, Data area 0	
Entries: CLEAR, FERROR	
Module FILENO	Stream to file descriptor conversion
Program 5, Data area 0	
Entries: FILENO	
Module FOPENB	Stream open binary mode
Program 170, Data area 6	
Entries: FOPENB	
Externals: E.O, FOPENA, H.	
Module FOPENA	Stream open ASCII mode
Program 297, Data area 10	
Entries: FOPENA	
Externals: C.GT, C.LT, C.NOT, C.TST, E., E.O, FOPEN, H., IOMODE, MAXCHN, SEEKEN	
Module SEEKEN	Seek to end of file, used for APPEND
Program 235, Data area 8	
Entries: SEEKEN	
Externals: C.UGT, CCBDOS, E.O, FFB, H., IOBIN, IOBUF, IOEND, IOFCB, IOIND, IOSECT, IOSIZE, Q.	
Module FREAD	Unix-style fread()
Program 114, Data area 8	
Entries: FREAD	
Externals: C.GT, C.NOT, GETC H.	
Module FSEEK	Unix-style fseek(), long integer seek()
Program 145, Data area 4	
Entries: FSEEK	
Externals: BL.HU, H., HI.BL, L.DIV, L.MOD, LLONG., Q., SEEK	
Module FTELLU	Unix-style ftell(), file position
Program 132, Data area 8	
Entries: FTELLU	
Externals: E.O, FTELL, FTELLR, H., HI.BL, HU.BL, L.ADD, L.MUL, LLONG., SLONG., ST4.	

## CLIBU.REL CONTENTS

Module FWRITE	Unix-style fwrite()
Program 108, Data area 8	
Entries: FWRITE	
Externals: C.GT, H., PUTC	
Module GETL	Get a long integer from stream
Program 88, Data area 8	
Entries: GETL	
Externals: C.NOT, GETC, H., LLONG., ST4.	
Module GETPAS	Get password no-echo from user
Program 137, Data area 13	
Entries: GETPAS	
Externals: C.TST, CCBIOS, H., INDEX, PUTCHA, Q.	
Module GETPID	Get process ID, fake function
Program 4, Data area 0	
Entries: GETPID	
Module GETW	Get word from stream
Program 78, Data area 6	
Entries: GETW	
Externals: C.NOT, C.TST, GETC, H.	
Module HEAPS	Heap sort
Program 484, Data area 14	
Entries: HEAPS	
Externals: C.DIV, C.GE, C.GT, C.TST, E.O, H., Q.	
Module ISATTY	Is a TTY system call
Program 43, Data area 0	
Entries: ISATTY	
Externals: C.NOT, E.O, H.	
Module REALLO	Re-allocate, Unix-style
Program 110, Data area 2	
Entries: REALLO	
Externals: E., FREE, H., MALLOC, MOVEME	
Module MALLOC	Memory allocate
Program 505, Data area 14	
Entries: FREE, F RE L, MALLOC	
Externals: C.UGE, C.UGT, C.ULE, C.ULT, E., E.O, H., Q., S., SBRK	
Module MKTEMP	Make a temp file name from template
Program 128, Data area 4	
Entries: MKTEMP	
Externals: C.TST, H., ITOA, STRLEN	
Module PERROR	Print system error message
Program 450, Data area 0	
Entries: PERROR, SYS ER, SYS NE	
Externals: C.GT, CON, E., ERRNO, H., Q.	

## CLIBU.REL CONTENTS

Module PUTL	Put long integer out to stream
Program 75, Data area 4	
Entries: PUTL	
Externals: H., HI.BL, LLONG., PUTC	
Module PUTW	Put 16 bit word out to stream
Program 84, Data area 4	
Entries: PUTW	
Externals: C.TST, H., PUTC	
Module QSORT	Quick sort, Unix-style arguments
Program 906, Data area 15	
Entries: QSORT	
Externals: C.DIV, C.GE, C.GT, C.LE, C.LT, C.MULT, E.O, H., S.	
Module QUICK	Quick sort, alternate arguments
Program 676, Data area 10	
Entries: QUICK	
Externals: C.DIV, C.GE, C.GT, C.LE, C.LT, E.O, H., Q., S.	
Module READA	Read ASCII, Unix-style
Program 80, Data area 4	
Entries: READA	
Externals: C.GT, C.NOT, GETC, H.	
Module READB	Read binary, modeled after READA
Program 80, Data area 4	
Entries: READB	
Externals: C.GT, C.NOT, GETCBI, H.	
Module SETBUF	Set file buffer location
Program 71, Data area 0	
Entries: SETBUF	
Externals: C.UGE, C.ULE, E.O, H., IOBUF, MAXCHN, Q., REWIND	
Module SHELLS	Shell sort
Program 343, Data area 12	
Entries: SHELLS	
Externals: C.DIV, C.GE, C.GT, C.TST, H., Q., S.	
Module SIGNAL	Signal interrupt process, fake
Program 4, Data area 0	
Entries: SIGNAL	
Module SWAB	Swap bytes in buffer to buffer
Program 110, Data area 4	
Entries: SWAB	
Externals: C.GT, H., Q.	

## CLIBU.REL CONTENTS

Module SYSTEM	System call for CP/M 2.2
Program 334, Data area 6	
Entries: SYSTEM	
Externals: C.GT, C.NOT, C.TST, CON, E., E.O, EXIT, FCLOSE, FOPEN, H., Q., REVERS, STRLEN, WRITE	
Module TTYNAM	Get tty name
Program 82, Data area 0	
Entries: TTYNAM	
Externals: .SWITC, H.	
Module WRITEA	Write function ASCII, Unix-style
Program 75, Data area 4	
Entries: WRITEA	
Externals: C.GT, H., PUTC	
Module WRITEB	Write binary, modeled after WRITEA
Program 75, Data area 4	
Entries: WRITEB	
Externals: C.GT, H., PUTCBI	
Module GETATT	Get file attributes for CP/M
Program 131, Data area 4	
Entries: GETATT	
Externals: E., FINDFI, H.	
Module SETATT	Set file attributes for CP/M
Program 171, Data area 4	
Entries: SETATT	
Externals: A., C.TST, CCBDDOS, H., MAKEFC	
Module BSEARC	
Program 178, Data area 8	
Entries: BSEARC	
Externals: C.ASR, C.MULT, C.NOT, C.TST, H., Q., S.	
Module LSEARC	
Program 160, Data area 4	
Entries: LSEARC	
Externals: C.NOT, C.TST, H., MOVEME, Q.	
Module UNIXIO	
Program 15, Data area 0	
Entries: ACCT, ALARM, CUSERI, DUP, DUP2, ENDGRE, ENDPWE, FORK, FSTAT, FTIME, GETEGI, GETEUI, GETGRE, GETGRG, GETGRN, GETLOG, GETPGR, GETPID, GETPPI, GETPW, GETPWE, GETPWN, GETPWU, GETUID, IOCTL, KILL, LINK, , MKNOD, MONITO, MOUNT, NICE, PAUSE, PCLOSE, PIPE, POPEN, PROFIL, PTRACE, PUTPWE, SETGID, SETGRE, SETPGR, SETPWE, SETUID, SLEEP, STAT, STIME, SYNC, TIMES, UMASK, UMOUNT, UNAME, USTAT, UTIME, WAIT	

## CLIBT.REL CONTENTS

Module RAND	Random number generator
Program 267, Data area 118	
Entries: RAND, RAND1, SRAND	
Externals: C.MULT, H., Q.	
Module ACOT	Arc cotangent
Program 29, Data area 0	
Entries: ACOT	
Externals: ATAN, F.ADD, F.NEG, LLONG.	
Module ACOS	Arc cosine
Program 168, Data area 4	
Entries: ACOS	
Externals: ATAN, CF.EQ, CF.GT, ERRNO, F.DIV, F.MUL, F.NEG, F.SUB, FABS, HI.BF, LLONG., SLONG., SQRT, SWAP4.	
Module ACSC	Arc cosecant
Program 162, Data area 4	
Entries: ACSC	
Externals: ASEC, CF.EQ, CF.LE, CF.LT, ERRNO F.DIV, F.MUL, F.NEG, F.SUB, LLONG., SLONG., SQRT, SWAP4.	
Module ASIN	Arc sine
Program 160, Data area 4	
Entries: ASIN	
Externals: ATAN, CF.EQ, CF.GT, ERRNO, F.DIV, F.MUL, F.NEG, F.SUB, HI.BF, LLONG., SLONG., SQRT, SWAP4.	
Module ASEC	Arc secant
Program 128, Data area 4	
Entries: ASEC	
Externals: ATAN, CF.LE, CF.LT, ERRNO F.MUL F.SUB LLONG. SLONG., SQRT, SWAP4.	
Module ATAN2	Arc tangent of y/x
Program 122, Data area 0	
Entries: ATAN2	
Externals: ATAN, CF.EQ, CF.LT, ERRNO, F.DIV, F.NEG, LLONG.	
Module ATAN	Arc tangent
Program 321, Data area 12	
Entries: ATAN	
Externals: CF.EQ, CF.GT, CF.LT, ERRNO F.DIV, F.MUL, F.NEG, F.SUB, HORNER, LLONG., SLONG.	
Module LDEXP	Load exponent
Program 31, Data area 0	
Entries: LDEXP	
Externals: H., LLONG.	

## CLIBT.REL CONTENTS

Module FREXP	Fraction exponent
Program 52, Data area 0	
Entries: FREXP	
Externals: A., H., LLONG., Q., S.	
Module TAN	Tangent
Program 458, Data area 10	
Entries: TAN	
Externals: ABS, BF.HI, C.TST, CF.EQ, CF.GT, COS, ERRNO, F.ADD, F.DIV, F.MUL, F.NEG, F.SUB, HORNER, I..F, LLONG., SIN, SLONG., SWAP4.	
Module COS	Cosine
Program 26, Data area 0	
Entries: COS	
Externals: F.ADD, LLONG., SIN	
Module SIN	Sine
Program 357, Data area 12	
Entries: SIN	
Externals: .SWITC, BF.HI, CF.LE F.DIV, F.MUL F.NEG F.SUB, HI.BF, HORNER, I..F, LLONG., SLONG., SWAP4.	
Module SQRT	Square root
Program 392, Data area 14	
Entries: SQRT	
Externals: CF.EQ, CF.GT, CF.LT ERRNO, F.ADD, F.DIV, F.MUL, HI.BF, HORNER, LLONG., SLONG.	
Module SINH	Hyperbolic sine
Program 122, Data area 0	
Entries: SINH	
Externals: CF.GT, CF.LT, ERRNO, EXP, F.DIV, F.NEG, F.SUB, LLONG.	
Module COSH	Hyperbolic cosine
Program 89, Data area 0	
Entries: COSH	
Externals: CF.GT, ERRNO, EXP, F.ADD, F.DIV, F.NEG, FABS, LLONG.	
Module TANH	Hyperbolic tangent
Program 159, Data area 4	
Entries: TANH	
Externals: CF.GT, CF.LT, ERRNO, EXP F.ADD F.DIV, F.NEG, F.SUB, LLONG., SLONG., SWAP4.	
Module EXP	Natural base exponential
Program 97, Data area 0	
Entries: EXP	
Externals: CF.GT, CF.LT, ERRNO, F.MUL, F.NEG, LLONG., POW10	

## CLIBT.REL CONTENTS

Module POW	Power function x to the y
Program 521, Data area 14	
Entries: POW	
Externals: BF.BL, BF.HI, BL.BF, C.GT, CF.EQ, CF.GT, CF.LT, E.O, ERRNO, F.DIV, F.MUL, F.NEG, FABS, G.HI.BF HI.BL L.AND, LLONG., LOG10, LONG.O, POW10, SLONG.	
Module POW10	Power function 10 to the y
Program 534, Data area 10	
Entries: POW10	
Externals: BF.HI, C.DIV, C.TST, CF.GT, CF.LE, CF.LT F.DIV F.MUL, F.NEG, F.SUB, FLT.O, HI.BF, HORNER, I..F, LLONG., SLONG.	
Module LOG	Natural logarithm base e
Program 26, Data area 0	
Entries: LOG	
Externals: F.MUL, LLONG., LOG10	
Module LOG10	Standard logarithm base 10
Program 370, Data area 22	
Entries: LOG10	
Externals: A., CF.EQ, CF.GE, ERRNO, F.ADD, F.DIV, F.MUL, F.NEG, F.SUB, HORNER, I..F, LLONG., S., SLONG., SWAP4.	
Module CEIL	Long integer ceiling
Program 77, Data area 4	
Entries: CEIL	
Externals: BF.BL, BL.BF, CF.LT, HI.BL, L.ADD, LLONG., SLONG.	
Module FLOOR	Long integer floor
Program 125, Data area 4	
Entries: FLOOR	
Externals: BF.BL, BL.BF, CF.GT, CL.LT, HI.BL, L.ADD, L.SUB, LLONG., SLONG.	
Module FMOD	Float remainder function
Program 99, Data area 4	
Entries: FMOD	
Externals: BF.BL, BL.BF, CF.EQ, ERRNO, F.DIV, F.MUL, F.SUB, LLONG., SLONG., SWAP4.	
Module MODF	Remainder function for integer part
Program 103, Data area 6	
Entries: MODF	
Externals: BF.HI, CF.LT, ERRNO, F.SUB, FABS, H., HI.BF, I..F, LLONG., Q., SLONG., SWAP4.	
Module RAD	Degrees to radians
Program 19, Data area 0	
Entries: RAD	
Externals: F.MUL, LLONG.	

## CLIBT.REL CONTENTS

Module DEG	Radians to degrees
Program 19, Data area 0	
Entries: DEG	
Externals: F.MUL, LLONG.	
Module LABS	Long absolute value
Program 42, Data area 0	
Entries: LABS	
Externals: CL.GE, HI.BL, L.NEG, LLONG.	
Module FABS	Float absolute value
Program 42, Data area 0	
Entries: FABS	
Externals: CF.GE, F.NEG, HI.BF, LLONG.	
Module ABS	Integer absolute value
Program 17, Data area 0	
Entries: ABS	
Externals: C.LE, C.NEG	
Module HORNER	Horners method for polynomial evaluation
Program 104, Data area 4	
Entries: HORNER	
Externals: F.ADD, F.MUL, H., LLONG., Q., SLONG.	

## INDEX TO THE USER GUIDE

#asm,#define,#endasm,#endif ... 8  
#if,#ifdef,#ifndef,#include,#undef ... 5,15  
\$\$\$.SUB ... 5  
\$88## ... 39  
\$LM ... 27,42  
\$OFF&X ... 42  
\$OFF1,\$OFF2,\$OFF3,\$OFF4,\$OFF5,\$OFF6 ... 26,27  
\$PCH## ... 39  
\$SIZ\$, \$SIZ&X, \$SIZ1, \$SIZ2, \$SIZ3, \$SIZ4, \$SIZ5, \$SIZ6 ... 26,27,42  
\$TAG&X, \$SIZ&X, \$OFF&X ... 42  
\$TAG1,\$TAG2,\$TAG3,\$TAG4,\$TAG5,\$TAG6 ... 26,27  
%f,%ld ... 15  
'\n' ... 20  
(FLOAT)1.570796326795,(FLOAT)3.14159265359 ... 21  
-Abackup.arc ... 24  
.PRINTX (M80)... 40,42  
.REQUEST (L80)... 16,40  
2-drive ... 1  
2.203 ... 3  
3.14159 ... 8  
3.44 ... 2  
3.45 ... 2,4  
320K ... 3  
32767 ... 8  
4096 ... 7,10,11,14  
4MHz ... 7  
<GRP.H> ... 36  
<lib1>,<lib2>,<lib3> ... 42  
<PWD.H> ... 36,37  
<SETJMP.H> ... 20  
<STDIO.H> ... 4,5,6,7,9,10,11,12,13,14,15,20,31,33  
<SYS/STAT.H> ... 34,35  
<SYS/TIMEB.H> ... 34,35  
<SYS/TYPES.H> ... 34,35,36  
<UNIX.H> ... 18,33  
<VG.H> ... 13  
a:\$\$\$.sub ... 5  
alarm(seconds) ... 33  
arc.com ... 1,24,25  
archive ... 1,23,24,25  
assembler ... 1,3,29  
atof ... 17,41  
atoi() ... 17  
atol ... 17,41  
auto-request ... 9  
automatically ... 9,12,13,15,16,24,27  
a exit ... 27,39  
back-up ... 24  
background ... 30  
Banahan ... 33  
BDOS ... 6,10,42  
Bourne's ... 6

# INDEX TO THE USER GUIDE

buffers ... 7,9,11,14,15,26,27,31  
bufszl ... 7,11,14,42  
BUGS ... 2  
c.com ... 1,2,3  
C/80 ... 1,2,3,4,8,11,12,16,17,18,19,27,30,31,41,42  
calloc() ... 18  
cbuf ... 27,31  
cc.com ... 1,2,3  
CCMAIN.REL ... 27,42  
cconfig.com ... 1,2,3  
ccp ... 10,14,32  
ccspace ... 10,11,14,31,32,39,42  
Cexit () ... 10,40  
cfreeT() ... 18  
chain ... 10,14,17,41  
character-only ... 39  
chario ... 6,14,39  
CL.COM ... 1  
CLIB.REL ... 1,2,22,26,27,28,39  
CLIB/L ... 28  
CLIB/U ... 27  
CLIBM ... 1,9,12,13,15,17,23,27,39  
CLIBMO.REL,CLIBM1.REL ... 1  
CLIBT ... 1,9,17,21,22,23,28  
CLIBT.ARC ... 28  
CLIBU ... 1,9,17,18,22,33  
CLIBX ... 1,9,17,27  
CLINK ... 1,4,5,12,13  
Cmode ... 27  
cmp.c ... 2  
compilation ... 4  
con() ... 6  
configuration ... 1,3,31  
conout(),cono 1(),cono 2 ... 16,39  
console ... 6,7,16,24,32,39  
conversion ... 3,6,18  
Copen ... 40  
copier ... 1,3,7  
cp ... 1,2,3,5,10,14,24,31,32,33,41  
CP/M-86 ... 5  
CPU-dependent ... 8  
CtlB ... 27  
Ctrl-a,Ctrl-c,Ctrl-j,Ctrl-z ... 20,24  
CTYPE.H ... 1  
cycle ... 12  
C mcln ... 40  
DUT.COM ... 2  
debugging ... 6,12  
December-1981 ... 2  
DecSystem ... 16  
devices ... 7,8,19,26  
diary ... 13

## INDEX TO THE USER GUIDE

direct-disk ... 41  
double ... 28  
DRI ... 1  
driver ... 16  
dsort,dsort16 ... 17,41  
dual-register ... 29  
dup(old),dup2(old,new) ... 34  
E2BIG,EACCES,EAGAIN,EBADF,EBUSY,ECHILD ... 22  
echo.c ... 7  
EDOM,EEXIST,EFAULT,EFBIG,EINTR,EINVAL ... 22  
EIO,EISDIR,EMFILE,EMLINK ... 22  
end-of-program ... 27  
END.REL ... 27  
endgrent() ... 36  
endpwent() ... 37  
ENFILE,ENODEV,ENOENT,ENOEXEC,ENOMEM ... 22  
ENOSPC,ENOTBLK,ENOTDIR,ENOTTY ... 22  
env1[1],env2[1] ... 20  
ENXIO ... 22  
EOF ... 7,16,39  
EPERM,EPIPE ... 22  
ERANGE ... 21,22  
Eratosthenes ... 14  
EROFS ... 22  
errno.h ... 1,22  
error(),err 1(),err 2 ... 4,39  
ESPIPE,ESRCH,ETXTBSY ... 22  
EX.COM ... 1  
EXDEV ... 22  
exit() ... 11,20  
exitto(addr) ... 10,30  
exp ... 23  
EXPLARGE ... 21  
extern ... 10,11,12,13,22,26,31  
EX SPC,TOT SZ ... 27,42  
f(T ... 13)  
FALSE ... 14,39  
fdexp() ... 28  
FDOS ... 9,10,31,32  
fgetc() ... 18,19  
filecopy ... 7  
fin,fin () ... 14,27  
fixed-origin ... 27  
flag ... 2,7,17,40  
FLIBRARY.REL ... 1  
float ... 2,9,15,17,21,23,28,29,39,40,41  
floating-point ... 15  
flush ... 14  
fopen(),fopena(),fopenb() ... 18  
force-searched ... 27  
fork() ... 34  
FORLIB.REL ... 27

## INDEX TO THE USER GUIDE

formats ... 3,15  
FORTRAN ... 16,27  
fout,fout\_() ... 14,27  
fprintf ... 9,31,39  
fprt 1(),fprt 2 ... 39  
fputs() ... 6  
free ... 4,9,14,18,26  
freopa,freopb(),freopen ... 18  
frexp() ... 28  
fscanf ... 9,15,39  
fseek ... 17,18,41  
fstat(fd,buf) ... 34  
ftell(),ftellu() ... 17,18,41  
ftime(tp) ... 34  
ftoa ... 17,41  
g() ... 13  
getc() ... 18,19  
getcbinary(fp) ... 20  
getegid() ... 34  
geteuid() ... 34  
getgid() ... 34  
getl ... 17,41  
getpgrp() ... 34  
getpid() ... 34  
getppid() ... 34  
getpw(uid,buf) ... 36  
gets() ... 6,16  
getuid() ... 34  
global ... 12,25,28,29,33  
go.com ... 1,5  
graphics ... 6,16  
h() ... 13  
h## ... 30  
H37 ... 3  
H89,Z90 ... 1,3,7,41  
handle ... 2,24,39  
heap ... 15,26,27,31,32  
HELLO.C ... 4,5,6,7  
hooks ... 9,17,18  
IF1,IF2,IFDEF,IFNDEF ... 40,42  
incremental ... 12,13  
indirection ... 32  
insert ... 13,17,18,40,41,42  
INTEL ... 25,28  
IObuf[fd] ... 26,32  
IOch[] ... 26  
ioctl(fd,request,argp) ... 34  
IOfcb[fd] ... 26,32  
IOfcb[],IObuf[] ... 26,31  
IOnchx,I0size ... 26  
IOTABLE ... 2,15,26  
IPIby180 ... 21

# INDEX TO THE USER GUIDE

IRP ... 40,42  
itoa() ... 17  
jmpbuffer ... 20  
jmp buf ... 20  
K&R,Kernighan ... 4,5,17  
kill(pid,sig) ... 34  
L80 ... 1,2,5,12,17,25,27  
landmarks ... 2  
LIB.COM ... 1,26,27  
lib1,lib2,lib3 ... 9,12,13,16,27,42  
LIBC.ARC,LIBCC.ARC ... 25,26  
librarian ... 1  
libraries ... 9,12,17,25,27  
library-dependent ... 8  
link(path1,path2) ... 34  
LINK-80 ... 4  
linker ... 1,2,12  
LLONG ... 17,41  
ln ... 21  
loaders ... 10  
log ... 21,23  
long/float ... 2  
longjmp(env,10) ... 20  
lseek() ... 18  
ltoa((long)x,buffer) ... 13,17,41  
M80 ... 1,2,14,17,40,42  
macro ... 3,8,16,17,18,19,26,40,42  
Macro-80 ... 3  
MAKEWORK.SUB ... 1,3  
malloc() ... 18  
manager ... 1,26  
MARC.COM ... 1,24  
MATH.H,MATHDEF.H ... 1,17,21  
mathlib ... 15,42  
MathPack ... 2,17,23  
MAX2EXP ... 21  
MAXCHN,MAXCHN-1 ... 15,26  
memmap.c ... 31,32  
Microsoft ... 1,3,4,25,26,27,40  
mknod(path,mode,dev) ... 34  
MLIB ... 17,42  
modules ... 12,13,14,15,25,27  
monitor(lowpc,highpc,buffer,bufsize,nfunc) ... 33,37  
mount(spec,dir,rwflag) ... 34  
multi-argument ... 9,39  
multiply-defined ... 33  
newlines ... 16  
nfiles ... 7,11,15,26,39,42  
nice(incr) ... 35  
non-float ... 9  
NULL ... 16,39,40  
nullcmd ... 5,6,11,14,40

# INDEX TO THE USER GUIDE

numsort ... 17,41  
obsolete ... 18  
open(),opena(),openb() ... 15,18,19,26  
optimize ... 40  
overlay ... 10,11  
passwd ... 35,36,37  
pause() ... 35  
PC ... 2  
pclose(fp) ... 35  
peekl ... 17,41  
peekw(1) ... 10,11  
peekw(6) ... 31  
performance ... 7,15  
perror() ... 22  
pfFLOAT,pfLONG ... 9,15,39,40  
PF OP1,PF OP2 ... 40  
PIby180,PIbyTwo ... 21  
PIP ... 1,3,24  
pipe(fd) ... 35  
pokeI ... 17,41  
port ... 18  
portability ... 23  
POW10INF ... 21  
pre-processor ... 8,9,11,14,16  
printf ... 4,5,6,9,13,15,16,28,39,40,41  
printf,prnt 1(),prnt 2,prnt 3,prnt 4 ... 39  
profil(buff,bufsize,offset,scale) ... 35  
profile ... 2  
printf,prtf 1(),prtf 2 ... 39  
ptrace(request, pid,addr,data) ... 35  
putc(c,stdout) ... 8  
putchar() ... 6,7,8,16  
putl ... 17,41  
putpwent(p,fp) ... 35  
puts ... 6,10,11,16,20,31,39  
puts(p+4096+i+strlen("keyword")) ... 11  
p\_fin,p\_fout ... 31  
quad ... 7  
rand1,random ... 17,41  
rdDISK ... 17,41  
re-direction ... 7,9,10,11,14,40  
re-open ... 18  
read,reada,readb ... 11,14,18,19,21,41  
REL ... 2,12,25  
response ... 7  
rewind() ... 26  
Ritchie ... 4  
RN## ... 40  
ROM,ROM-based,romable ... 3,16,25  
run("overlay") ... 10  
Rutter ... 33  
sample=clib<sample>/e ... 28

# INDEX TO THE USER GUIDE

sbinary ... 17,41  
scanf() ... 9,15,39,40,41  
searches ... 12,13,17  
seek ... 17,41  
Sep-1982 ... 2  
setbuf() ... 26  
setgid(gid) ... 35  
setgrent() ... 36  
setjmp(),setjmp.h,longjmp ... 1,20  
SETLIB ... 17,40,41  
setpgrp() ... 35  
setpwent() ... 37  
setuid(uid) ... 35  
sfFLOAT,sfLONG ... 9,15,40  
sfree() ... 26,27  
SF\_OP1,SF\_OP2,SF\_OP3 ... 40  
sgtty ... 34  
sieve ... 14  
signal ... 1  
SIGNAL.H ... 1  
sleep(seconds) ... 37  
SLONG ... 17,41  
soft-sectored ... 7  
SPHL ... 10,30  
sprintf,sprt\_1(),sprt\_2 ... 9,39  
sscanf ... 9,15,39  
ssort2,ssort3 ... 17,41  
stack ... 10,14,29,30,31,32  
stat(name,buf) ... 35  
stderr,stdin,stdout ... 10,14,16,39  
stime(tp) ... 35  
STK\_POS,f scan,scan f,s scan ... 17,39  
stream ... 7,16,19,26,39,41  
style ... 11  
SUBMIT.COM ... 1  
subroutines ... 9,14  
switch(setjmp(env1)),switch(setjmp(env2)) ... 20  
switches ... 9,11,12,14,15,16,17,26,39,40  
symbol ... 3,8,9,17,21,25,27,40  
SYMBOLS.ARC ... 25  
SYMBOLS.SYM ... 25  
sync() ... 35  
sysgen,sysgend ... 1,3  
test1.c,test2.c ... 7  
time(tloc) ... 35  
timer ... 17,41  
times(buffer) ... 35  
TLIB ... 17,21,41,42  
tokens ... 40  
tolower ... 18  
Toolworks ... 1,2,4,17,19,23  
TOT\_SZ ... 27,42

## INDEX TO THE USER GUIDE

toupper ... 18  
TRACE ... 2  
transcendental ... 1,23,42  
TREE.C ... 2  
trial,trial/n/y/e ... 25  
trial.c,trial.sym ... 25  
trick ... 5  
TRUE ... 1,6,8,14,39  
TWOBYPI ... 21  
type ... 4,12,13,16,28,39  
type,type\_1(),type\_2,type\_3 ... 39  
typedef ... 8  
uid ... 35,36,37  
ULIB ... 17,42  
umask(cmask) ... 36  
umount(spec) ... 36  
uname(name) ... 36  
unbuff(fd) ... 26  
Unix ... 1,6,14,15,16,17,18,19,22,23,33,39,41,42  
Unix-compatible ... 39  
Unix-standard ... 15  
UNIX.H ... 1,18,19  
UNIXIO.H,UNIXIO.REL ... 33  
uppercase ... 14  
USELIB ... 17,40,42  
ustat(dev,buf) ... 36  
utime(path,times) ... 36  
V,V1,V2,V3,CLIBM/S,VG,CLIB/S,V/n/e ... 12  
V.C,V.COM,V1,V2,V3,VG.C,VG.H,VG.REL ... 12,13,42  
void ... 34,35,39  
wait(2),wait(x) ... 33  
wait(stat loc) ... 36  
waitz,wafEz(500\*x) ... 17,33,41  
wildcards ... 24  
Wiley ... 33  
winchester ... 7  
wrDISK ... 17,41  
write,writea,writeb ... 5,14,18,19,22,24,41  
x,<1,2,3,4,5,6> ... 42  
XDIR.COM,XDIRUTIL.COM ... 1  
XLIB ... 17,42  
XREFSYM.C,XREFSYM.COM ... 25  
Z90/Z100 ... 41  
  exit, exit() ... 14,39  
\_tolower,\_toupper ... 18

**C LIBRARY VERSION 3.2**

**Library by:** G.B.Gustafson  
2243 South Belaire Drive  
Salt Lake City, UT 84109  
1-801-466-6820

**Compiler:** C/80, Version 2.0/3.0/3.1,  
by The Software Toolworks

**Assembler:** M80.COM, by Microsoft

**Linker:** L80.COM, by Microsoft

**Librarian:** LIB.COM, by Microsoft

**Copyright 1985 by G.B.Gustafson and Viking C Systems. All rights reserved.**

**Trademarks.** CP/M, CP/M-68K, DDT, SID are trademarks of Digital Research, Incorporated. Unix, Unix V7, Unix V5, Unix V3, PCC are trademarks of Bell Laboratories, Incorporated. C/80, MathPack, Flibrary, PIE are trademarks of The Software Toolworks. MBASIC, FORLIB, M80, L80, LIB, F80, BASCOM, BASLIB, MS-DOS are trademarks of Microsoft, Incorporated.

# IMPATIENT USER'S GUIDE

**MAKING A WORK DISK.** Format and sysgen a work disk for drive A:, and copy these files onto the new disk, in the order given. Files marked with asterisk (\*) are optional. Files marked with a pound sign (#) are utilities with a specific function - replace the name given with the one you normally use.

XDIR.COM	*	Sorted directory program (source XDIRUTIL.COM)
GO.COM		Made with SAVE 0 GO.COM, 0 records.
CC.COM		Software Toolworks Compiler (See notes)
M80.COM		Microsoft assembler (See notes)
L80.COM		Microsoft linker (See notes)
PI.COM	*	A text editor for source files
PIP.COM	*	DRI CP/M 2.2 copier for moving files
CP.COM	*	Copier for moving files
CLINK.COM		C compile & assemble & link
EX.COM	*	Memory submit program.
CL.COM	*	C compile & assemble & link, H89/Z90.
STDIO.H		Standard I/O header file
UNIX.H	*	Unix header file for CLIBU.REL
ERRNO.H	*	Error reporting header file for CLIBT.REL
MATH.H	*	Math package header file for CLIBT.REL
CTYPE.H	*	Array-based character class header file
SETJMP.H	*	Setjmp/longjmp header file
SIGNAL.H	*	Signal header file
CLIB.REL		Main library
CLIBX.REL		Extensions of the main library
CLIBM.REL	*	Math library, floats and longs (see notes)
CLIBU.REL	*	Unix library
CLIBT.REL	*	Transcendental function library
LIB.COM	*	Microsoft librarian
STAT.COM	*	DRI CP/M 2.2 STAT utility
ARC.COM	*	Archive manager, part I
MARC.COM	*	Archive manager, part II
CCONFIG.COM	*	C/80 Configuration for CC.COM

If you have a 2-drive system, then MAKEWORK.SUB and SUBMIT.COM may be used to do the work. This method is recommended if you are familiar with SUBMIT. If not, then ignore these remarks and use PIP to copy the files.

## Notes :

- o To make CC.COM from C.COM on the C/80 3.1 distribution disk, run the program CCONFIG.COM and set the M80 switch to TRUE. Save the file as CC.COM.
- o MAKING CLIBM.REL FROM CLIBM0.REL AND FLIBRARY.REL

```
A>PIP CLIBM0.REL=B:CLIBM0.REL  
A>PIP CLIBM1.REL=B:FLIBRARY.REL  
A>LIB CLIBM=CLIBM0,CLIBM1/E  
A>ERA CLIBM0.REL  
A>ERA CLIBM1.REL
```

## IMPATIENT USER'S GUIDE

- o The best version of M80 is December-1981, 3.44. Others cannot handle lowercase, or have bugs when used with submit utilities.
- o The best version of L80 is Sep-1982, 3.45. Earlier versions cannot link some of the REL files in CLIB.REL. The problem is in external arithmetic. The only cure is a good linker. To understand the problem, look at IOTABLE.CC in the sources.
- o Some version 3.0 distribution disks had a file CCONFIG.COM that omitted long and float code generation options. The correct CCONFIG for floats and longs appeared on the MATHPACK disk.
- o Both version 2.0 and 3.1 compilers generate source code that is compatible with this library. Some of the C/80 sources are not duplicated in this package, notably the PROFILE and TRACE sources and sample programs like CMP.C and TREE.C.
- o Compilers have bugs. If you find one that doesn't, let everyone know. We'll buy it! If your C/80 compiler is more than a few months old, send \$12.00 to The Software Toolworks for an update. The growth of the long/float compiler had its landmarks - get the latest version.
- o **MAKING CC.COM FROM C.COM under C/80 version 2.0**  
The file CC.COM is made from C.COM on the C/80 distribution disk by patching the M80-compatibility flag, using the CP/M program DDT.COM. Here are the steps for **VERSION 2.0 ONLY:**

```
A>DDT C.COM
      DDT VERS 2.2
      NEXT PC
      7C00 0100
      -s0143
      0143 00 1
      0144 00 .
      -g0
A>save 123 CC.COM
```

# I M P A T I E N T   U S E R ' S   G U I D E

## o MAKING CC.COM FROM C.COM under C/80 version 3.0+

The file cc.com is a copy of c.com on the distribution disk, made using PIP. Following the copy, run CCONFIG.COM, and set the variables as below. This configuration works for CP/M 2.203 on an H89 with 64k memory, H37 drives. Other configurations will require reductions in the sizes for A, B, D.

Programs with large string use may need to set A, D near 0 and increase B to 3000 or more.

---

### C/80 CONFIGURATION:

A: Symbol table size (-s): 370  
B: String constant table size (-c): 1000  
C: Dump constants after each routine (-k): YES  
D: Macro (#define) table size (-d): 2100  
E: Switch table size (-w): 200  
F: Structure table size (-r): 400  
G: Merge duplicate string constants (-f): NO  
H: Assembler (-m): Microsoft Macro-80  
I: Initialize arrays to zero (-z): < 256 BYTES ONLY  
J: Generate ROMable code in Macro-80 (CSEG/DSEG) (-a): YES  
K: Screen size (for error msg pause; 0 for none) (-e): 0  
L: Generate slightly larger, faster code (-o): YES  
M: Sign extension on char to int conversion (-x): NO  
N: Device for library files (-v): A:

## o Contents of MAKEWORK.SUB. The submit file used to make a work disk assumes that drive A: contains a sysgened and formatted disk of size 320k or larger. Drive B: is to contain all the source files listed below. The service of the operation is to place the files on the work disk in optimal order. The file copier cp is supplied with the package.

B:CP A:=B:mdir.com,go.com,cc.com,m80.com,l80.com  
B:CP A:=B:pi.com,PIP.COM,CP.COM,CLINK.COM,EX.COM,CL.COM  
B:CP A:=B:stdio.h,unix.h,errno.h,math.h,ctype.h,setjmp.h,signal.h  
B:CP A:=B:clib.rel,clibx.rel,clibm.rel,clibu.rel,clibt.rel  
B:CP A:=B:lib.com,stat.com,arc.com,marc.com,cconfig.com

## TUTORIAL ON COMPILER OPERATION

Kernighan & Ritchie HELLO WORLD. To create, compile, assemble and link the classic Kernighan and Ritchie program HELLO.C, start your text editor and type these lines:

```
#include <stdio.h>
main()
{
    printf("Hello World!\n");
}
```

COMPILE, ASSEMBLE, LINK. The process is handled completely by CLINK.COM, as follows:

A>CLINK HELLO

The result of CLINK is a SUBMIT run that prints the following:

A>clink hello

```
A>ERA HELLO.COM
NO FILE
A>CC HELLO-HELLO
```

C/80 Compiler 3.1 (3/10/84) - (c) 1984 The Software Toolworks

0 errors in compilation
1K bytes free

```
A>MB0 +HELLO
* STDIO.H - Ver 3.2 *
* CLIB Request *
```

No Fatal error(s)

```
A>ERA HELLO.MAC
A>L80 HELLO,HELLO/N/E
```

Link-80 3.45 26-Sep-82 Copyright (c) 1981 Microsoft

Data 0103 0AB6 < 2483>

40072 Bytes Free
[0133 0AB6 10]

A>ERA HELLO.REL

The program HELLO.COM should run as follows:

```
A>HELLO
Hello World!
```

A>

# TUTORIAL ON COMPILER OPERATION

**O P T I O N A L   R U N   M E T H O D .** An optional method for running your program exists just after L80 finishes operation. This method requires that you have on disk a file called **GO.COM** created as follows:

```
A>SAVE 0 GO.COM
```

To start HELLO.COM just after L80 finishes the link, enter go as below to produce:

```
A>go  
Hello World!
```

```
A>
```

This trick works with unbanked CP/M 2.2 only. Don't try it under CP/M-86 or CP/M 3.0. It can be repeated as many times as desired. The file GO.COM causes a jump to address 100 hex, which runs the currently loaded program.

**S A V I N G   C L I N K   C O M M A N D S .** CLINK creates a file A:\$\$\$.SUB with the desired commands. The lines of this command file are printed during execution. The contents of \$\$\$.SUB may be saved in HELLO.SUB with this command line:

```
A>CLINK HELLO -S
```

**C O N T R O L   O F   O B J E C T   C O D E   S I Z E .** The main objection of assembly language programmers to language C is the unreasonable overhead attributed to the C library. Some implementations cause the minimum executable file to be 16k bytes or larger.

In contrast, this library lets you link in just what you need to make the program run. As an example, here is how to write the K&R HELLO.C source in order to make it compile into an executable file of less than 256 bytes.

```
#define nullcmd 1
#include <stdio.h>
#define putchar con
#undef printf
printf(p) char *p; { while(*p) putchar(*p++); }
main()
{
    printf("Hello world\r\n");
}
```

## TUTORIAL ON COMPILER OPERATION

**A N E V E N S M A L L E R H E L L O . C O M .** The above is not the smallest HELLO.COM possible. It is possible to reduce the size even further, as follows:

```
#define nullcmd 1
#include <stdio.h>
#define putchar con
main()
{
    static char *p;
    p = "Hello world\r\n";
    while(*p) putchar(*p++);
}
```

The library function con() is a raw-mode function that prints characters to the console via a standard BIOS call. It is useful for debugging, graphics and conversion of assembly language code segments to C source files. It is the same function that an assembly language programmer would use to print to the console.

The same idea can be used with the library printf(), because it calls putchar(). Here is a re-coding of HELLO.C that compiles to less than 1024 bytes, yet still includes the entire printf support code.

```
#define nullcmd 1
#include <stdio.h>
putchar(x) int x; { con(x);}
main()
{
    printf("Hello world!\r\n");
}
```

**U S I N G P U T S .** The well-known Unix functions gets() (get a string from the console) and puts() (put a string to the console) are included, true to the description in Bourne's book *The Unix System*. In particular, gets() eats the cr/lf on input and puts() appends a cr/lf on output. Here is HELLO.C coded with puts():

```
#define nullcmd 1
#define chario 1
#include <stdio.h>
main() { puts("Hello world!");}
```

**U S I N G F P U T S .** The overhead of printf() can be avoided entirely by coding the C program with fputs(), provided that no numerical conversions are required. One such program is HELLO.C:

```
#define nullcmd 1
#define chario 1
#include <stdio.h>
main() { fputs("Hello world!\n", stderr);}
```

## TUTORIAL ON COMPILER OPERATION

**ECHO EXAMPLE.** As a final example, consider the standard program echo. It is a disguised version of filecopy, due to the possibility of I/O re-direction.

```
#define C80 1
#define C80
#define nfiles 2      /* 2 pre-defined file buffers */
#define bufsz1 2048    /* stream 1 buffer is 2k   */
#define bufsz2 2048    /* stream 2 buffer is 2k   */
#endif
#include <stdio.h>
main()
{
static int x;
    while( (x = getchar()) != EOF) putchar(x);
}
```

If run with a standard command line, then each line typed at the console will appear twice, making for a not-so-interesting program. However, use of the re-direction flags > and < turns ECHO.C into a file copier.

To illustrate,

```
A>echo <test1.c >test2.c
```

copies input file test1.c to output file test2.c. Experiment with the idea, using devices 1st: and con:, to obtain a print program and a console listing program from echo.c.

**IMPROVING DISK PERFORMANCE.** Settable buffer size makes for dramatic improvement in the response time of programs like ECHO. Experiment with ECHO.C using different buffer sizes. Optimization depends on the features of your disk controller and disk drives.

Drives rated at 96 tpi respond well with buffer size 4096. Ram disk does just as well with buffer size 128. Winchester disk does best with buffer size 1024, due to its internal buffering.

Compiler operation on an H89/Z90 machine at 4mhz is optimal with Quad soft-sectorized disks using 1024 sectors (790k per disk). The best performance occurs with files copied in the order outlined above. In this case, HELLO.C will compile in 54 seconds. Other disk configurations and file orders can increase the compile time for HELLO.C by as much as 25 seconds (1 minute and 20 seconds total).

## P R E - P R O C E S S O R   D I R E C T I V E S

The pre-processor used by C/80 does not support code macros, e.g.,

```
#define putchar(c) putc(c,stdout)
```

will not compile under C/80.

C/80 does not support `typedef`, but the pre-processor may be used to get around `typedef` problems.

The most commonly used pre-processor directives are:

```
#define  
#ifdef  
#ifndef  
#if  
#endif  
#asm  
#endasm  
#include
```

### P r e - P r o c e s s o r   A p p l i c a t i o n   N o t e s :

- o The more pre-processor directives used, the less portable will be your code across all compilers.
- o Across full C compilers, the reverse is true. For this sub-class of compilers, the pre-processor directives help to overcome problems with different CPU devices, addressing schemes and library interfaces.
- o Use a `#define` directive whenever it is expected that the code being written is CPU-dependent, library-dependent or it uses information about the host system that could fail for another compiler.
- o Commonly used constants like 3.14159 or 32767 should be supplied with a symbol, via `#define`.
- o Array sizes and constant loop bounds should be given as a `#define`, in order to adjust usage uniformly across the source code.
- o A `#define` need not contain constants. You may put variable names in the definition, even if they have not yet been defined in the source. The real restriction is usage: don't try to use something that has not been defined.

The header file STDIO.H contains many pre-processor directives and assembly language hooks. Its main job is:

1. Compile a module as MAIN or as a subroutine.
2. Auto-request libraries CLIBX, CLIBT, CLIBU, CLIBM.
3. Request user libraries through the lib1, lib2, lib3 pre-processor directives.
4. Switch long and float libraries on and off.
5. Set up multiple-argument functions like printf().
6. Turn on/off command line arguments and re-direction.
7. Set the amount of free space under FDOS.
8. Set the number of file buffers and their size.

The Version 3.2 library automatically determines the main program by checking for the appearance of the reserved symbol MAIN. If main() is not in the MAC file made by C80, then the file is compiled as a collection of subroutines.

By default, usage of printf and scanf select the non-float and non-long versions, in order to reduce the object code size.

To use float and/or long versions of printf and scanf, you must include switches in your source code, as follows:

```
#define pfFLOAT 1      /* Use floats in printf() */
#define pfLONG 1       /* Use longs in printf() */
#define sfFLOAT 1      /* Use floats in scanf() */
#define sfLONG 1       /* Use longs in scanf() */
```

These switches must be known to the compiler by the time it reads STDIO.H. The same switches control code size for the functions fprintf(), sprintf(), fscanf(), sscanf().

The example below enables floats for printf() and longs for scanf(). There is a code size savings realized by not linking in irrelevant support code (e.g., we don't need longs in printf()).

```
#define pfFLOAT 1
#define sfLONG 1
#include <stdio.h>
main()
{
    float f;
    long L;
    while((scanf("%ld",&L)) > 0) {
        f = L;
        printf("f = %6.4f\n",f);
    }
}
```

## S T D I O . H H E A D E R - O v e r l a y s

Headroom under FDOS can be reserved, for the purpose of building loaders or to allow chained variables. In particular, you can leave the CP/M CCP resident, and exit to the CCP instead of to WARM BOOT.

Below, we illustrate how to avoid the exit to warm boot, and how to chain variables to an overlay. The method used avoids problems with re-direction: the stdin and stdout streams are closed on exit.

```
#define ccospace 2048      /* standard CP/M CCP size */
#include <stdio.h>
main()
{
extern char *CC CCP;
    puts("Leaving MAIN, exit to CCP");
    exitto(CC_CCP);
}

exitto(address)
char *address;
{
    Cexit_();
    address;      /* load address of caller's stack */
#asm
    SPHL          /* load stack pointer, pop and jump */
#endifasm
}
```

The second use of ccospace is to communicate variables to an overlay. The idea is to set up a common block in the area between the C stack and BDOS. The overlay should check a keyword area to insure that the linkage is valid. Here's an example:

```
#define ccospace 4096      /* lots of room */
#include <stdio.h>
main()                  /* program EXAMPLE.COM links to OVERLAY.COM */
{
char *p,*peekw();
    p = peekw(1);      /* address of BDOS */
    strcpy(p-4096,"KeyWord");
    strcat(p-4096,"This the data in the common block");
    puts("Main running, running overlay.com");
    run("overlay");
}
```

## STDIO.H HEADER - Style

```
#define ccspac 4096
#include <stdio.h>
main()                                /* this is OVERLAY.COM */
{
char *p,*peekw();
extern char *CC CCP;
    p = peekw(1);
    puts("Overlay running");
    if(strcmp(p-4096,"KeyWord") == 0) {
        puts(p-4096+1+strlen("KeyWord")); /* display data */
    }
    else {
        puts("Bad overlay linkage"); exit();
    }
}
```

Style dictates that you isolate compiler and library dependent source code from the rest of your program by the use of pre-processor #define switches. Here is what we suggest:

```
#define C80 1          /* C/80 & Library 3.2 switch */

#if C80
#define nullcmd 1      /* Turn off re-direction */
#define nfiles 3       /* Three file buffers */
#define bufsz1 4096    /* first buffer size 4k */
#endif

#include <stdio.h>      /* Now read in STDIO.H */
```

## STUDIO.H HEADER - Libraries

Incremental development is recommended for programs that exceed 300 lines or 20k in source. While the C/80 compiler can compile large programs directly, the compile time will not be acceptable for debugging sessions. Use of libraries and subroutine modules can decrease the compile, assemble, link cycle time to a few minutes.

The scheme used for incremental development is to place all globals in one file, then invent a header file for use in other modules. The header file contains type declarations and extern definitions.

Modules that use the globals will #include the header file for the global variables. Even main() itself will #include the global header file. In this way, related sets of routines can be collected to a single file, compiled once into a REL file, and used thereafter as a library.

Searches of libraries and the search order can be controlled by the STUDIO.H #define switches 1ib1, 1ib2, 1ib3.

For example, suppose that the main program is called V.C and is to be linked with modules V1.C, V2.C, V3.C, VG.C.

Further, assume that V3.C requires CLIBM.REL during link, but V.C does not. Then V.C should be set up as follows:

```
#define 1ib1 V1,V2,V3,CLIBM,VG  
#include <stdio.h>  
main()  
{  
    ... code as usual  
}
```

The command line given to make V.COM is CLINK V. All linkage is handled automatically, in the order specified, which is equivalent to:

```
L80 V,V1,V2,V3,CLIBM/S,VG,CLIB/S,V/N/E
```

It is important to note that modules V1, V2, V3, VG are assumed to be compiled separately with V1.REL, V2.REL, V3.REL, VG.REL on the default disk. The linker L80 looks on the default drive for libraries. Sometimes you can omit the CLIBM reference because it already exists in the file V3.REL (due to using STUDIO.H).

**E x a m p l e .** Sample incremental development.

Below you will see a complete diary of a sample program V.COM, created by incremental development via modules V1, V2, V3, VG, V.

```

A>type V1.C
#include <stdio.h>
f()
{
    printf("Function f\n");
}

A>type V2.C
#include <stdio.h>
g()
{
    fputs("Function g\n",stderr);
}

A>type V3.C
#include <vg.h>
#include <stdio.h>
char *h()
{
    ltoa((long)x,buffer);
}

A>type VG.C
/*
 * VG.C
 * globals used by V,V1,V2,V3
 */
char buffer[21];
long x = 3546091;

A>type VG.H
/*
 * VG.H
 * globals used by V.C, V1.C, V2.C, V3.C
 */
extern char buffer[21];
extern long x;

A>type V.C
#define libl V1,V2,V3,CLIBM,VG
#include <stdio.h>
main()
{
    f();
    g();
    h();
}

```

**A>clink v** - This compiles and links the program automatically.

This example may work with CLIBM omitted from the list, because STDIO.H inserts a library request into V3.REL, causing a search of the library CLIBM.REL. However, there is no sure way to control the search order except by the method above.

Data statements cannot force a library search. But executable ones that use the reserved symbols in STDIO.H will cause a search. The usual error made by programmers is the omission of #include <stdio.h> or incremental compiles that do not document the library searches back in main().

## STDIO.H DEFINITIONS

The file STDIO.H is to be used with M80 assemblies of subroutines and the main program. The following assembly switches are used. Declarations are preprocessor #define directives, which must appear before STDIO.H is read into the source code. We use **boldface** and lowercase to distinguish the user-settable switches. Constants are all UPPERCASE.

<b>stderr</b>	The Bell Labs Unix symbols are defined in header file STDIO.H to use usual defaults: stderr=252, stdin=fin, stdout=fout. The latter have been implemented as function returns from fin_() and fout_().
<b>stdin</b>	
<b>stdout</b>	
<b>TRUE</b>	Both TRUE and FALSE are defined as in the standard set by Bell Lab's Portable C Compiler (PCC). FALSE = 0, TRUE = 1.
<b>FALSE</b>	
<b>_exit</b>	The standard abort exit. Does not flush file buffers. Does not close the file control block to the disk directory. Leading underlines cause trouble with M80/L80, hence the pre-processor solution that you see here.
<b>bufsz1</b>	Settable buffer sizes began with version 3.0. Three switches are provided in STDIO.H. The default is 256. It is suggested that you use size 4096 for fast I/O on sequential files. For rapid seeks, use size 128 or 256. Usage: #define bufsz1 4096, before #include <stdio.h>.
<b>bufsz2</b>	
<b>bufsz3</b>	
<b>chario</b>	Use #define chario 1 before #include <stdio.h>. Most of the fat for a C object file comes from the file support package and re-direction modules. This switch unloads the fat. Recommended for programs like the Sieve of Eratosthenes, which simply prints to the screen.
<b>ccspace</b>	The space under CP/M can be adjusted at compile time via the switch #define ccspace nnn. Choose nnn = 1024 to get 1k of free space, etc. This feature is useful if you wish to leave the CCP resident and write your own exit routine. It also provides for a method of chaining variables without stack collision.
<b>nullcmd</b>	Unload the command line package with #define nullcmd 1. Use it before the #include <stdio.h> in your source file.

## STDIO.H DEFINITIONS

MODULES	A module is any C function or source file that does not contain main(). In versions 3.1 or earlier a module declaration was needed. This has been automated in version 3.2 by declaring it a module if the reserved word main does not appear.
<b>mathlib</b>	The switch for a library request to CLIBM.REL is #define mathlib 1, used before #include <stdio.h>. Note that the printf and scanf float & long switches will automatically request CLIBM.REL as needed. Further, any explicit use of long or float will cause CLIBM to be searched.
<b>sffLOAT</b>	Turns on the float support for scanf(). Usage: #define sffLOAT 1, before the normal #include <stdio.h>, enables floating-point code for scanf, fscanf, sscanf.
<b>sflong</b>	Turns on the long support for scanf(). If neither are selected but your code contains float and long references, then formats like %ld and %f will be processed but not acted upon. The only warning issued is the unexpected performance.
<b>pffloat</b>	Turns on the float support for printf(). Switches the printf() definitions from the main library version to the portable Unix-standard version. Usage: Before #include <stdio.h>, enter the line #define pffloat 1.
<b>pflong</b>	Same function as pffloat only for long integer support. You may use either or both or none. Using neither causes the main library printf to be used. To force use of the portable printf, #include the proper header file (but #undef printf first!). Usage: #define pflong 1.
<b>nfiles</b>	The number of files that may be open simultaneously is a constant MAXCHN in the module IOTABLE.REL. To change it from the usual 6, alter and re-assemble IOTABLE. To change the number of I/O buffers from the usual 3, use #define nfiles nnn where nnn is any number from 0 to 6. The number of file buffers and their size affects the total heap space available to the program.

## STDIO.H DEFINITIONS

EOF	The Unix standard of -1 for end of file is used throughout the library.
NULL	The usual value of 0 is used for NULL. The cast of this variable is always (int) unless otherwise changed.
FILE	The Unix system uses pointers for stream I/O whereas C/80 uses the older version 5 system standard of channel numbers. To make them compatible in source, define FILE to be (int) as in STDIO.H and use FILE *chan; rather than int chan; in your source code.
lib1 lib2 lib3	There are three switches provided: lib1, lib2, lib3. These variables are actually C/80 macros used by the pre-processor. The effect is to put .REQUEST lib1 in the source code, etc. You may use any string for the macro replacement (C/80 version 3.0 has a bug, but 3.1 and 2.0 work as expected.). Usage: #define lib1 mylibrary.
puts	The version of puts() in the main library meets the Unix standard. It supports multiple arguments.
type	The function called type() is modeled after the function of the same name used in Decsystem Fortran. It types out strings using putchar(). It differs from puts() in that newlines are not output automatically. This function is defined by a MACRO in stdio.h - you may change the name TYPE to whatever seems appropriate.
error	The function called error() is the same as type(), except that it always writes to the console. It is a replacement for printf in ROM-based applications.
conout	The conout() code supports multiple-argument printing to the console only. It is used primarily as a device driver for screen graphics, in which the delivery rate must be as rapid as possible.
gets	Unix standard get-string from stdin. Strips the linefeed from the string so that puts() will output the same thing that you type.

## STDIO.H DEFINITIONS

ltoa	Converts long integer to ascii text, similar to itoa(). Located in CLIBM.REL with library select done by STDIO.H. See K&R.
atol	Converts ascii text to a long integer, similar to atoi(). Located in CLIBM.REL with library select done by STDIO.H. See K&R. Object code from the C/80 Mathpack.
ftoa	Converts float to ascii text. Provided with the C/80 MATHPACK from The Software Toolworks. Not a K&R function - this one uses four arguments, not two. Located in CLIBM.REL.
LIB	A macro definition for M80, which inserts library request information into the MAC file.
SETLIB	A macro definition for M80, which sets a flag for use of a particular library. Used to auto-select libraries based on reserved symbol names.
USELIB	A macro definition for M80, which invokes a library request for L80.
ULIB TLIB MLIB XLIB	These assembly constants are used as switches to invoke searches of libraries CLIBU, CLIBT, CLIBM and CLIBX, respectively. See also the header files MATH.H, UNIX.H.

The following keywords are used to set up hooks to the libraries:

SLONG., LLONG., seek, wrDISK, rdDISK, timer, numsort, ssort3, ssort2, dsort, dsort16, waitz, binary, sbinary, run, chain, insert, STK\_pos, pokel, peekl, fseek, ftellu, getl, putl, randl, ltoa, atof, atol

## UNIX.H DEFINITIONS

The header file UNIX.H contains the following definitions.

```
#define _tolower    tolower      /* C/80 has no argument macros */
#define _toupper    toupper
#define Fopen        fopena      /* Ascii open */
#define freopen      freopa      /* Ascii re-open */
#define ftell        ftellu      /* Unix ftell */
#define lseek        fseek       /* happen to match in C/80 */
#define read         reada       /* ascii read */
#define write        writea      /* ascii write */
#define fgetc        getc        /* both are functions */
#define open         opena       /* ascii open */
#define cfree        free        /* cfree for calloc */
```

In addition, it inserts into your program a library request for the library CLIBU.REL.

_toupper	These conversion macros are not possible under C/80, so we map them to genuine function calls.
_tolower	
fopen	The honest function is fopen(). We also include fopenb() for binary I/O, but name fopen() is assigned to fopen(). Note that the C/80 name fopen() does not conflict. The only possible disaster is in leaving off #include <UNIX.H>, which drops all the hooks. Source code compiled with UNIX.H will port to most Unix V7 systems without change.
freopen	The normal freopen is ASCII mode. To get binary mode you have to use freopenb(), which does not have a portable Unix V7 analog.
ftell	Past naming conventions of the C/80 library use ftell() with an integer return. This ftell() has a standard Unix V7 long integer return. The actual function used is ftellu(), but to keep the source code looking like Unix V7, please use ftell().
lseek	It so happens that lseek() exists in the library, by accident. This is an obsolete Unix function, but it may be handy to know how to access it under this library.
cfree	The cfree() function paired to calloc() is identical to the free() function used with malloc().

## U N I X . H D E F I N I T I O N S

read	The naming conventions of C/80 again conflict with a standard Unix V7 interface. The problem is that the Software Toolworks functions operate in fixed binary mode. They are incapable of using devices and will not translate carriage return and linefeed combinations. Worst, there is no real end of file detection. To cure the problem, we created functions reada(), readb(), writea(), writeb() which do real stream I/O. In UNIX.H, we assign read to reada and write to writea. Code written with UNIX.H uses standard names that transport easily to all full C systems.
fgetc	The function fgetc() is identical to getc(). Under most systems, getc() is a macro definition. But C/80 does not support such things, so getc() and fgetc() are one and the same.
open	The honest functions are opena() and openb(), which do ASCII and BINARY opens, respectively. We name open() as opena() to be Unix V7 compatible.

## SETJMP.H DEFINITIONS

The header file SETJMP.H contains the following essential lines, which MUST be used if your source code calls either setjmp() or longjmp().

```
struct jmpbuffer { char *STACKP; char *RETADR;};
#define jmp_buf struct jmpbuffer
```

Recall that setjmp and longjmp act as a pair to back out of deeply buried function calls. Suppose, for example, in the process of opening up data files it is found at some level that a data file is corrupt. The function pair setjmp and longjmp give an elegant way to back out of the mess, close the data files and start over.

To make it clear how the pair setjmp() and longjmp() interact in source, we reproduce here the example found in SETJMP.H.

Example: Test program for setjmp() and longjmp().

```
#include <stdio.h>
#include <setjmp.h>
jmp_buf env1[1];
jmp_buf env2[1];

f(fp)
FILE *fp;
{ int x;
x = getcbinary(fp); puts("");
switch(x) {
case 'Z'-'@': longjmp(env1,10);
case '\n' : longjmp(env1,20);
case 'C'-'@': longjmp(env1,30);
case 'A'-'@': longjmp(env2,1);
}
}

main()
{
puts("Setup for env1");
switch(setjmp(env1)) {
case 0: break;
case 10: puts("EOF encountered on input");
case 20: puts("Linefeed in input"); break;
case 30: puts("Ctrl-C normal exit"); exit();
default: puts("Undefined error");
}
puts("Setup for env2");
switch(setjmp(env2)) {
case 0: break;
case 1: puts("ctrl-A struck"); break;
default: puts("Undefined error");
}
puts("Special characters are ctrl-A, ctrl-C, ctrl-J, ctrl-Z");
while(TRUE) f(stdin);
}
```

## MATH.H DEFINITIONS

Below is the contents of MATH.H. Please take note that MATHDEF.H is different from MATH.H. The former is used to compile the library CLIBT.REL.

```
#define FLOAT      float
#define PI          (FLOAT)3.14159265359
#define PIbyTWO    (FLOAT)1.570796326795
#define TWObyPI    (FLOAT)0.63661977
#define ONE         (FLOAT)1.0
#define ZERO        (FLOAT)0.0
#define INF         (FLOAT)1.0e39
#define EXPLARGE   (FLOAT)89.80081863
#define POW10INF   (FLOAT)39.0
#define MAX2EXP    129
#define PIby180     (FLOAT)0.0174532925
#define IPIby180   (FLOAT)57.29577951
#define EDOM        33
#define ERANGE      34
#define ln          log
#endif
#define TLIB SET 1
#endif
```

The purpose of the symbol TLIB is to provide linkage to the library CLIBT.REL. This is done by a switch in STDIO.H, therefore it is essential that MATH.H be read into your source file prior to STDIO.H.

## ERRNO.H DEFINITIONS

Below are the error codes normally supported under Unix V7 from Bell Labs.

### The external variable

```
extern int errno;
```

should be declared in any source code that uses the error reporting features of the library. The variable itself is located in CLIB.REL. You may put it in your source code if desired, thereby omitting the library variable during link.

At present, error reports are supported in the library CLIBT.REL only. To print an error, you may use the function perror() in CLIBU.REL, or else write your own.

The error codes that are supported appear in perror.c in the source archives. We refer you to that source plus any standard reference on the Bell Labs Unix System III error codes. A printed copy of perror.c appears in the library manual.

```
/*
 * errno.h - error codes
 */
#define EPERM    1          #define EFBIG   27
#define ENOENT   2          #define ENOSPC  28
#define ESRCH    3          #define ESPIPE  29
#define EINTR    4          #define EROFS   30
#define EIO      5          #define EMLINK  31
#define ENXIO    6          #define EPIPE   32
#define E2BIG    7
#define ENOEXEC  8          /* math bound checks */
#define EBADF   9
#define ECHILD  10
#define EAGAIN  11
#define ENOMEM  12
#define EACCES  13
#defineEFAULT 14
#define ENOTBLK 15
#define EBUSY   16
#define EEXIST  17
#define EXDEV   18
#define ENODEV  19
#define ENOTDIR 20
#define EISDIR  21
#define EINVAL 22
#define ENFILE  23
#define EMFILE  24
#define ENOTTY  25
#define ETXTBSY 26
```

## GENERATION OF CLIBT

The sources for CLIBT.REL are in the transcendental function archive. All header files required are included. Please note that this library uses special properties of The Software Toolworks MathPack. There is no claim made about the portability of the code. Unix code assumes doubles, and this package uses float only.

The transcendental function library is built from the following command line sequence. Ordering of functions is essential, because the references in the library must resolve in one pass. In addition, a library search of CLIBM.REL is required to pick up the needed parts of the long and float library.

```
B:  
A:LIB  
CLIBT=  
RAND  
ACOT  
ACOS  
ACSC  
ASIN  
ASEC  
ATAN2  
ATAN  
LDEXP  
FREXP  
TAN  
COS  
SIN  
SQRT  
SINH  
COSH  
TANH  
EXP  
POW  
POW10  
LOG  
LOG10  
CEIL  
FLOOR  
FMOD  
MOOF  
RAD  
DEG  
LABS  
FABS  
ABS  
HORNER  
/E
```

## L I B R A R I A N   A R C   &   M A R C

L I B R A R Y   S O U R C E   M A N A G E R . The main features of ARC.COM are as follows:

- o Several versions of the same source can be maintained. Files are distinguished by their physical location in the archive, the date, and finally the actual contents.
- o Archives may be combined with PIP to make a master archive. Archives are text files terminated with ctrl-Z.
- o Single files can be copied out of the archive. This includes paged printing with settable left margin.
- o Archives can be split into smaller ones.
- o Files within an archive may be selectively deleted.

The functions of adding files to an archive versus the management of the archive have been separated. ARC.COM handles an archive after creation. MARC.COM creates archives and adds files. Following are the functions of MARC:

- o Adding files to an archive uses wildcards from the command line. The speed is similar to PIP, but you can view the file and add notes. Files in the list may be archived or skipped over.
- o Many small files can be combined into an archive to make better use of disk space. For example, limitations of 64 directory entries disappear, because the archive uses full blocks.
- o The principal reason why MARC exists is to back-up on disk several versions of a C program during development. You don't change the name, just the date and the notes. Usage:

```
A>marc myfile.c "<-d<today's date>" -aBACKUP.ARC -c
```

The best and easiest implementation uses a CP/M submit file.

ARC is capable of producing neat listings on the printer. Just write to the LST: device in SINGLE COPY mode. Options of LEFT MARGIN and PAGE LENGTH can be set, for automatically paging the output and allowing for ring-binder holes. Output to LST: ends with a formfeed.

ARC will tag module names taken from a source file. The result is that files in the archive are tagged, just as though you had typed a tag command T at the console for each file. This feature is very useful for collecting source files into one file.

A sample run of ARC and MARC appears in an appendix. We refer the reader to those pages for detailed information on running the programs.

## P O R T A B L E   S O U R C E   G E N E R A T O R   X R E F S Y M

**P O R T A B L E   S O U R C E   C O D E   G E N E R A T O R .** The generator creates source code files for programs that access the libraries used by L80. In theory, you can generate the complete source required for a ROM application.

The generator is a two-step operation which filters the symbol file from L80 through XREFSYM.COM and finally through ARC.COM to produce the portable source archive.

The purpose of the symbol cross-reference generator called XREFSYM.C is to generate a cross-reference table which maps the global symbols in the REL file to the source code modules in the archives LIBC.ARC and LIBCC.ARC. The information needed by XREFSYM.COM resides in the archive SYMBOLS.ARC and the Intel Symbol file produced by Microsoft's L80.

To operate XREFSYM.COM, for a source file TRIAL.C, do these steps:

```
A>CC TRIAL  
A>M80 =TRIAL  
A>L80 TRIAL,TRIAL/N/Y/E  
A>ERA TRIAL.REL  
A>XREFSYM TRIAL.SYM
```

The output is SYMBOLS.SYM, which is used by ARC.COM to tag the relevant files, and make a complete portable source file for the program TRIAL.C.

The condition and readability of the source code that is generated is a function of the source library and its contents. It is suggested that one maintain a C - only source library, and forget about trying to transfer the massive I/O library. Generally, files with extension .CC are not portable, however exceptions are likely.

A run of XREFSYM is given in an appendix, along with more detail on usage. Generally, XREFSYM is used intermittently. It is little use to casual users.

## FILES, BUFFERS, ASSEMBLY CONSTANTS

**S E T T A B L E   B U F F E R   S I Z E S .** New with version 3.0 of the library is the option of settable buffer size and the possibility of dynamic re-assignment of buffers. The changes to the library tightened error checking and added three entries into the I/O table (namely, IOsize, IOnhx, IOend) - see IOTABLE.CC in LIBCC.ARC.

Generally, you should set the buffer size and the number of files by means of macro switches in STDIO.H. However, it is possible to dynamically alter the buffer size and buffer location at run time. This is safe only in case you precede such changes with a rewind() function call on the open stream. See setbuf(), rewind() and the examples therein.

The fixed initial size of each buffer is defined in STDIO.H by the assembly constants \$SIZ1, ..., \$SIZ6 . Two other constants \$TAG1 and \$OFF1 are generated for use in IOTABLE.REL (see STDIO.H). The default size is 256.

The number of files is fixed at 6. Devices CON:, RDR:, PUN:, LST: are recognized but are not part of the file count. They use device numbers 252, 253, 254, 255. Device 0 is mapped to device 252.

Up to 251 files may be in simultaneous use, provided the module IOTABLE.REL is assembled to reselect the correct number of file buffers. See IOTABLE.CC for re-assembly instructions and STDIO.H for header file declarations.

To replace re-assembled IOTABLE.REL in the main library CLIB.REL, use this command line for the Microsoft library manager LIB.COM:

```
A>LIB NEWLIB=CLIB<..IOTABLE>,IOTABLE,CLIB<IOTABLE+1..>/E  
A>ERA CLIB.REL  
A>REN CLIB.REL=NEWLIB.REL
```

The file buffers and file control blocks for each of the files selected with the nfiles assembly switch (#define nfiles 6, for example) reside physically in memory just after the program data and text.

File buffers and file control blocks built at run time in excess of the nfiles count (but not more than MAXCHN-1 are possible) will call sbrk() to get more space. The sfree() function that frees up heap space obtained from sbrk() will not free up file buffer areas. A fragmented heap can result. The latter can cause program failure due to lack of contiguous heap space.

To manually throw away the buffers assigned to file descriptor fd, use this function (not portable and potentially dangerous):

```
unbuff(fd) int fd;  
{  
extern char IOch[]; extern int IOfcb[],IObuf[];  
if(IOch[fd] == -1) IOfcb[fd] = IObuf[fd] = 0;  
}
```

## L I B R A R I E S , M A I N T E N A N C E & R E B U I L D S

While the above throws away the buffers, sbrk() will not forget until you call sfree() to reset the end-of-program pointer. Use sfree(-1) to get rid of all the assigned heap space. The latter includes everything sbrk() assigned up to the call to sfree.

**F O R C E - L O A D I N G O F R E L F I L E S .** The file CLIB.REL has the same use as FORLIB.REL in FORTRAN. The STDIO.H file causes modules compiled by C/80 to undergo a library search of CLIB.REL on exit from L80.

Modules other than CLIB.REL can be force-searched by using the STDIO.H symbols lib1, lib2, and lib3. The search order is the order of appearance, with final search of CLIBX, CLIBM, CLIB, in that order, as required.

You will usually not have to force the search of the libraries. The header file STDIO.H causes most of the search requests to be processed automatically.

**L I B R A R Y I N T E G R I T Y C H E C K .** Below is a sample run of LIB.COM to verify forward references in the library CLIB.REL (use the /U - switch). CLIB.REL must start with its fixed-origin routine CCMAIN.REL and end with END.REL. The latter is used by storage allocation, therefore it must be the last module linked by L80. Your customized library passes the test if it reproduces the sample run.

```
A>LIB  
*CLIB/U  
Unsatisfied external request(s):  
$LM    A EXIT CBUF  CMODE   CTLB.   EXIT    EX SPC  FIN  
FOUT   MAIN   TOT SZ $OFF1   $OFF2   $OFF3   $OFF4   $OFF5  
$OFF6   $SIZ1   $SIZ2   $SIZ3   $SIZ4   $SIZ5   $SIZ6   $TAG1  
$TAG2   $TAG3   $TAG4   $TAG5   $TAG6  
*^C  
A> end sample run
```

```
Symbol generation  
YOUR PROGRAM: MAIN  
CCMAIN.REL: $LM A EXIT CBUF CMODE CTLB. EXIT FIN FOUT  
STDIO.H:   EX SPC TOT SZ  
           $OFF1 $OFF2 $OFF3 $OFF4 $OFF5 $OFF6  
           $SIZ1 $SIZ2 $SIZ3 $SIZ4 $SIZ5 $SIZ6  
           $TAG1 $TAG2 $TAG3 $TAG4 $TAG5 $TAG6
```

**M A I N T E N A N C E A N D R E - B U I L D S .** To replace module sample in CLIB.REL, use Microsoft's LIB.COM as follows:

```
A>LIB NEWLIB=CLIB<sample-1..>,sample,CLIB<sample+1..>/E  
A>ERA CLIB.REL  
A>REN CLIB.REL=NEWLIB.REL
```

## **DATA TYPES AND INTERNALS**

To extract a module from CLIB.REL:

```
A>LIB SAMPLE=CLIB<sample>/E
```

To list the global symbols in CLIB.REL:

```
A>LIB CLIB/L
```

**DATA TYPES.** The supported data types and their sizes are:

char	8 bits
short	16 bits
int	16 bits
unsigned	16 bits
long	32 bits
float	32 bits

There is no double data type. To convert existing programs that use doubles, or to develop new programs that will ultimately use doubles, place in your program:

```
#define double float
```

**LONG INTEGER FORMAT.** The format for long integers is reverse intel format, so that an integer is the lower 16 bits of a long integer. Long and integer pointers will point to the lowest byte. There is no unsigned long data type at present, however you may simulate it and print it out using printf().

**FLOAT EXPONENT RANGE.** Floats may have exponents in the range -38 to +38. The exponent format is 128-complement in the last Intel reverse format byte of the 32-bit word. See the sources in CLIBT.ARC for detailed information, especially fdexp() and frexp().

# ASSEMBLER INTERFACE FOR C/80

## CALLING CONVENTIONS - C FUNCTIONS.

The arguments for a function call are pushed onto the stack in the order listed. On return, the arguments are popped off the stack by the most efficient method.

The stack adjustment does not affect the return value, which is HL in the case of integer returns or a dual-register pair for long or float returns.

All stack pushes are 16 bits or 32 bits. Arguments that are long integer or float values cause a 32-bit push.

To unscramble a function call in assembly language, POP HL to obtain the return address. Each successive POP yanks one argument of width 16 bits, in the reverse order of the argument list. For example,

```
int x;
int f(a,b)
int a,b;
{
    x = a; x = b;
}
```

is written in assembler as:

```
DSEG
x:: DW 0          ;storage is global, in data segment
CSEG
f:: DS 0          ;function f is a global, in code segment
    POP HL        ;return address of caller
    POP DE        ;value of argument b
    POP BC        ;value of argument a
    PUSH BC
    PUSH DE
    PUSH HL        ;stack restored, ready to use DE,BC
    LXI HL,x
    MOV M,C
    INX H
    MOV M,B        ;x = a
    LXI HL,x
    MOV M,E
    INX H
    MOV M,D        ;x = b
    RET
```

## ASSEMBLER APPLICATION NOTES

There are some common tricks used in the C/80 assembly interface, often seen in C source code. One such is:

```
exitto(addr) char *addr;
{
    addr;
#asm
    SPHL
#endifasm
}
```

The purpose of the strange "addr;" in the function body is to load register pair HL with the address in the argument list.

The assembly code immediately following the statement loads the stack pointer with HL and the implied return effects a processor return from the stack at the aforementioned address. In 8080 code:

```
exitto:: DS 0
        LXI H,2      ;fetch addr off stack
        DAD SP
        CALL h.##    ;load HL indirect from HL
        SPHL        ;this is the assembly code you see above
        RET         ;ending brace is a RETURN
```

Such code as above is generally useless, unless you had the foresight to save the caller's stack frame and make your own stack. The possibility of corrupting the caller's stack still looms in the background, so we leave the issue with the programmer.

## MEMORY MAP

MEMORY LAYOUT UNDER CP/M 2.2. The standard C/80 configuration is altered somewhat in this library, to allow for dynamic file buffers. The test program MEMMAP.C illustrates how:

```
#include <stdio.h>
#define CPMBASE          0          /* cpm base */
#define TBUFFER          0x80        /* transient buffer */
#define CPMSTART          0x100       /* start address transient program */
main(argc,argv) int argc; char **argv;
{
int i;
auto int k;
extern int IOfcbl[],IObuf[];
extern char *Cbuf;
extern char *p_fin,*p_fout;
char *sbrk();
if(argc <= 1) {
    puts("MEMMAP displays the memory map of a C program");
    puts("including command line arguments and buffers.");
    puts("Usage: A>memmap arg1 arg2 ...");
    exit(0);
}
fprintf(stderr,"CP/M base = %04x Hex\n",CPMBASE);
fprintf(stderr,"CP/M default buffer = %04x Hex\n",TBUFFER);
fprintf(stderr,"CP/M program start address = %04x Hex\n",CPMSTART);

fprintf(stderr,
        "Physical end of code and data = %04x Hex\n",IOfcbl[1]);
for(i=1;i<=nfiles;++i) {
    fprintf(stderr,"IOfcbl[%d] = %04x Hex\n",i,IOfcbl[i]);
    fprintf(stderr,"IObuf[%d] = %04x Hex\n",i,IObuf[i]);
}

fprintf(stderr,"Heap starts at %04x Hex\n",sbrk(0));
fprintf(stderr,"Top of heap is at stack = %04x Hex\n",&k);
fprintf(stderr,"Token table starts at %04x Hex\n",&argv[0]);
fprintf(stderr,
        "Command line copy starts at %04x Hex\n",&argv[0][0]);
fprintf(stderr,"Console buffer starts at %04x Hex\n",Cbuf);
fprintf(stderr,"Console buffer ends at %04x Hex\n",Cbuf+140);
fprintf(stderr,"Base of CP/M FDOS is at %04x Hex\n",peekw(6));
fprintf(stderr,
        "C program upper limit is FDOS - ccspce = %04x Hex\n",
        peekw(6) - ccspce);
fprintf(stderr,"Indirection file name for >: %s\n",
        p_fout ? p_fout : "NONE SUPPLIED");
fprintf(stderr,"Indirection file name for <: %s\n",
        p_fin ? p_fin : "NONE SUPPLIED");
fprintf(stderr,"Program Name: %s\n",argv[0]);
for(i=1;i<argc;++i) {
    fprintf(stderr,"argv[%d] = %s\n",i,argv[i]);
}
}
```

## SAMPLE RUN OF MEMMAP.C

Here is a sample run. Note the mapping of lower to upper by the CCP and the handling of quoted characters.

```
A>memmap arg1 arg2 >foo <memmap.com "White space test" arg3
CP/M base = 0000 Hex
CP/M default buffer = 0080 Hex
CP/M program start address = 0100 Hex
Physical end of code and data = 106C Hex
I0fcb[1] = 106C Hex
I0buf[1] = 1090 Hex
I0fcb[2] = 1190 Hex
I0buf[2] = 11B4 Hex
I0fcb[3] = 12B4 Hex
I0buf[3] = 12D8 Hex
Heap starts at 1308 Hex
Top of heap is at stack = D5E9 Hex
Token table starts at D5F1 Hex
Command line copy starts at D635 Hex
Console buffer starts at D676 Hex
Console buffer ends at D702 Hex
Base of CP/M FDOS is at D706 Hex
C program upper limit is FD0S - ccspacE = D706 Hex
Indirection file name for >: FOO
Indirection file name for <: MEMMAP.COM
Program name: MEMMAP
argv[1] = ARG1
argv[2] = ARG2
argv[3] = WHITE SPACE TEST
argv[4] = ARG3
```

A>

## U N I X I O . H C O N T E N T S

```
/*
 * UNIXIO.H
 *
 * Purpose:
 *
 *      Provides stub functions for unix system calls and
 *      unix functions that have no analog under CP/M.
 *
 * Usage:
 *
 *      Copy the essential portions of this source file into
 *      your source code. Delete all unwanted lines.
 *
 * Example: The cure for MULTIPLY-DEFINED GLOBAL.
 *
 *      #include <unix.h>
 *      #include <stdio.h>
 *      monitor() { return -1; }
 *      main() {
 *          alarm(1);
 *          wait(2);
 *      }
 *      wait(x) int x;
 *      {
 *          waitz(500*x);
 *      }
 *
 *      The above code will end up with WAIT as a MULTIPLY-DEFINED
 *      GLOBAL. WAIT is already a global. ALARM selects the module
 *      UNIXIO.REL from CLIBU and hence a second instance of WAIT.
 *
 * The Cure:
 *
 *      Edit the source, pasting in as much of this file as required
 *      to resolve all symbols within the source files. To wit, add:
 *
 *          alarm() { return 0; }
 *
 *
 * Basic Reference: Banahan & Rutter, The Unix Book, Wiley Press,
 * New York (1983).
 *
 * Unix Reference: The Unix Programmers Manual
 *
 *
 * ===UNIX SYSTEM CALLS===
 *
 * int acct(file)           Return 0.
 * char *file;
 *
 * unsigned alarm(seconds)  Return 0.
```

## U N I X I O . H C O N T E N T S

```
* unsigned second;  
*  
* char *cuserid(s) Return (char *)0.  
* char *s;  
*  
* int dup(old) Return -1 for error.  
*  
* int dup2(old,new) Return -1 for error.  
*  
* int fork() Return -1 for error.  
*  
* #include <sys/types.h>  
* #include <sys/stat.h>  
* int fstat(fd,buf) Return -1 for error.  
* int fd;  
* struct stat *buf;  
*  
* #include <sys/types.h>  
* #include <sys/timeb.h>  
* void ftime(tp) Void return.  
* struct timeb *tp;  
*  
* int getpid() Return 0.  
*  
* int getuid() Return 0.  
*  
* int getgid() Return 0.  
*  
* int geteuid() Return 0.  
*  
* int getegid() Return 0.  
*  
* int ioctl(fd,request,argp) Return -1 for error.  
* int fd,request;  
* struct sgtty *argp;  
*  
* int getpgrp() Return 0.  
*  
* int getppid() Return 0.  
*  
* int kill(pid,sig) Return -1 for error.  
* int pid,sig;  
*  
* int link(path1,path2) Return -1 for error.  
* char *path1,*path2;  
*  
* int mknod(path,mode,dev) Return -1 for error.  
* char *path;  
* int mode,dev;  
*  
* int mount(spec,dir,rwflag) Return -1 for error.  
* char *spec,*dir;
```

## U N I X I O . H C O N T E N T S

```
* int rwflag;
*
* int nice(incr) Return 0.
* int incr;
*
* void pause() Void return.
*
* int pclose(fp) Return -1 for error.
* FILE *fp;
*
* int pipe(fd) Return -1 for error.
* int fd[2];
*
* FILE *popen(command,type) Return 0.
* char *command,*type;
*
* void profil(buff,bufsize,offset,scale) Void return.
* char *buff;
* int bufsize,offset,scale;
*
* int ptrace(request,pid,addr,data) Return -1 for error.
* int request,pid,addr,data;
*
* int putpwent(p,fp) Return -1 for error.
* struct passwd *p;
* FILE *fp;
*
* int setgid(gid) Return -1 for error.
* int gid;
*
* int setpgrp() Return 0.
*
* int setuid(uid) Return -1 for error.
* int uid;
*
* #include <sys/types.h>
* #include <sys/stat.h>
* int stat(name,buf) Return -1 for error.
* char *name;
* struct stat *buf;
*
* void sync() Void return.
*
* long time(tloc) Returns (long)0
* long *tloc;
*
* #include <sys/types.h>
* #include <sys/timeb.h>
* long times(buffer) Return -1 for error.
* struct tbuffer *buffer;
*
* int stime(tp) Return -1 for error.
```

## U N I X I O . H C O N T E N T S

```
* long *tp;
*
* int umask(cmask) Return 0.
* int cmask;
*
* int umount(spec) Return -1 for error.
* char *spec;
*
* int uname(name) Return 0.
* struct utsname *name;
*
* char *userid(s) Return (char *)0
* char *s;
*
* int ustat(dev,buf) Return -1 for error.
* int dev;
* struct ustat *buf;
*
* #include <sys/types.h>
* int utime(path,times) Return -1 for error.
* char *path;
* struct utimbuf *times; or time_t timep[2];
*
* int wait(stat_loc) Return -1 for error.
* int *stat_loc;
*
*
* ===UNIX MISCELLANY===
*
* #include <grp.h>
* struct group *getgrent() Return 0.
*
* #include <grp.h>
* struct group *getgrgid(gid) Return 0.
* int gid;
*
* #include <grp.h>
* struct group *getgrnam(name) Return 0.
* char *name;
*
* int setgrent() Return 0.
*
* int endgrent() Return 0.
*
* char *getlogin() Return (char *)0.
*
* int getpw(uid,buf) Return -1 for error.
* int uid;
* char *buf;
*
* #include <pwd.h>
* struct passwd *getpwuid(uid) Return 0.
```

# UNIXIO.H CONTENTS

```
* int uid;
*
* #include <pwd.h>
* struct passwd *getpwent()           Return 0.
*
* #include <pwd.h>
* struct passwd *getpwnam(name)       Return 0.
* char *name;
*
* int setpwent()                     Return 0.
*
* int endpwent()                     Return 0.
*
* int monitor(lowpc,highpc,buffer,bufsize,nfunc)   Return 0.
* int (*lowpc)();
* int (*highpc)();
* short buffer[];
* int bufsize;
* int nfunc;
*
* char *ttynname(fd)                 Return (char *)0.
* int fd;
*
* int sleep(seconds)                Return 0.
* unsigned seconds;
*/
#endif
#asm
times:: DS 0          /* long return */
        LXI B,0
        LXI D,0
        RET
acct::  DS 0
alarm:: DS 0
getpid:: DS 0
getuid:: DS 0
getgid:: DS 0
geteuid:: DS 0
getegid:: DS 0
getpgrp:: DS 0
getppid:: DS 0
nice::   DS 0
popen::  DS 0
setpgrp:: DS 0
umask::  DS 0
uname::  DS 0
getgrent::DS 0
getgrgid::DS 0
getgrnam::DS 0
setgrent::DS 0
endgrent::DS 0
getpwuid::DS 0
getpwent::DS 0
```

## U N I X I O . H C O N T E N T S

```
getpwnam:::DS 0
setpwent:::DS 0
endpwent:::DS 0
monitor::: DS 0
sleep::: DS 0
cuserid::: DS 0
getlogin:::DS 0
    LXI H,0
    RET
dup::: DS 0
dup2::: DS 0
fork::: DS 0
fstat::: DS 0
ioctl::: DS 0
kill::: DS 0
link::: DS 0
mknod::: DS 0
mount::: DS 0
pclose::: DS 0
pipe::: DS 0
ptrace::: DS 0
putpwent:::DS 0
setgid::: DS 0
setuid::: DS 0
stat::: DS 0
stime::: DS 0
umount::: DS 0
ustat::: DS 0
utime::: DS 0
wait::: DS 0
getpw::: DS 0
ftime::: DS 0
pause::: DS 0
profil::: DS 0
sync::: DS 0
    LXI H,-1
    RET
    END
```

#endasm

## STDIO.H CONTENTS

```
/*STDIO.H vers 3.2*/
#define EOF -1
#define NULL 0
#define TRUE 1
#define FALSE 0
#define FILE int
#define void int
#define VOID int
#define stdin (FILE *)fin_()
#define stdout (FILE *)fout_()
#define stderr (FILE *)252
#define _exit exit
#ifndef chario
putchar(){
#asm
    MVI A,252
    JMP $PCH#
#endifasm
}
getchar(){
#asm
    JMP $B8#
#endifasm
}
#endiff
#ifndef pfLONG
#ifndef pfFLOAT
#define printf prtf_1(),prtf_2
#define fprintf fpri_1(),fpri_2
#define sprintf spri_1(),spri_2
#endiff
#endiff
#ifndef printf
#define printf prnt_1(),prnt_2
#define fprintf prnt_1(),prnt_3
#define sprintf prnt_1(),prnt_4
#endiff
#define puts type_1(),type_3
#define type type_1(),type_2
#define error err_1(),err_2
#define conout cono_1(),cono_2
#define scanf STK_pos(),scan_f
#define sscanf STK_pos(),s_scan
#define fscanf STK_pos(),f_scan
#ifndef ccspace
#define ccspace 0
#endiff
#ifndef nfiles
#define nfiles 3
#endiff
#asm
```

End of file  
Unix NULL  
Unix TRUE  
Unix FALSE  
Unix-compatible FILE  
Unix-compatible void return

Std input stream  
Std output stream  
Std error reporting stream  
Hook \_exit() to real routine  
Set up character-only I/O

Console device handle  
Vector to console output.

Vector to console input.

Set up switches for printf.  
First one is fast, small integer  
only version in CLIB.REL, which  
has limited recursion facility.

Otherwise, use the portable  
printf in CLIBM, which has all  
features including long and float.

Define multi-argument puts().  
Define puts() without newline.  
Alternate type() for console only.  
Direct console output, multi-arg.  
Define scanf for multi-arguments.  
Note lack of recursion.

Define space above C program if  
not defined by user.

Define number of active files if  
not already defined by user. High  
number is 6.

# STDIO.H CONTENTS

```
#ifdef pfFLOAT
#ifndef pfLONG
EXT PF_OP1 ;long off
#endif
#endif
#ifndef pfLONG
#ifndef pfFLOAT
EXT PF_OP2 ;float off
#endif
#endif
#ifndef sfLONG
#ifndef sfFLOAT
EXT SF_OP1 ;long&float
#else
EXT SF_OP2 ;long only
#endif
#endif
#ifndef sfFLOAT
#ifndef sfLONG
EXT SF_OP3 ;float only
#endif
#endif
#endif
#define nullcmd
CSEG
IF1
.PRINTX * Re-defined Copen_, Cexit_, tokens, C_mcln *
ENDIF
Copen ::= DS 0
Cexit ::= DS 0
tokens ::= DS 0
C_mcln ::= DS 0
JMP RN$##
#endif
LIB MACRO Y
IRP X,<Y>
.REQUEST X
IF2
.PRINTX * X Request *
ENDIF
ENDIF
ENDIF
SETLIB MACRO X,Y
IFDEF X
Y SET 1
ENDIF
ENDIF
USELIB MACRO X,Y
IFDEF X
LIB Y
ENDIF
ENDIF
```

Set up portable printf switches for long and float code includes. Allows end-user programs with the portable printf and all excess bulk removed.

Set up scanf switches for long and float code includes. These switches allow end-user programs with long and/or float bulk removed to optimize code size.

Set up null command line options which get rid of re-direction bulk.

Make vacant routines to take the place of the real ones.

Return 0.

Define library request macro for Microsoft M80/L80.

Define set library macro for Microsoft M80/L80. Turns on a flag which tells us to request a library.

Define library use macro for Microsoft M80/L80. Inserts a library request if the symbol is defined.

## STDIO.H CONTENTS

SETLIB SLONG,,MLIB	Store long integer
SETLIB LLONG,,MLIB	Load long integer
SETLIB seek,XLIB	C/80 seek
SETLIB wrDISK,XLIB	Direct-disk read
SETLIB rdDISK,XLIB	Direct-disk write
SETLIB timer,XLIB	H89/Z90/Z100 timer
SETLIB numsort,XLIB	Distribution sort
SETLIB ssort3,XLIB	Shell sort, MBASIC rules
SETLIB ssort2,XLIB	Shell sort, array
SETLIB dsort,XLIB	Distribution sort
SETLIB dsort16,XLIB	Distribution sort, 16-bit
SETLIB waitz,XLIB	Wait for Z90/Z100
SETLIB binary,XLIB	Binary search, integer array
SETLIB sbinary,XLIB	Binary search, string array
SETLIB run,XLIB	Run CP/M command line
SETLIB chain,XLIB	Chain to tpa, fcb
SETLIB insert,XLIB	Insert CP/M command tail
SETLIB STK_pos,XLIB	Sccanf function
SETLIB poke1,MLIB	Poke long integer
SETLIB peek1,MLIB	Peek long integer
SETLIB fseek,MLIB	Long seek, Unix
SETLIB ftellu,MLIB	Long ftell, Unix
SETLIB get1,MLIB	Get long integer from stream
SETLIB put1,MLIB	Put long integer to stream
SETLIB rand1,TLIB	Random number primitive
SETLIB ftoa,MLIB	Float to ascii
SETLIB atof,MLIB	Ascii to float
SETLIB ltoa,MLIB	Long to ascii
SETLIB ato1,MLIB	Ascii to long
SETLIB prnt 1,MLIB	Portable printf function
SETLIB PF_OP1,MLIB	Printf option
SETLIB PF_OP2,MLIB	Printf option
SETLIB SF_OP1,MLIB	Sccanf option
SETLIB SF_OP2,MLIB	Sccanf option
SETLIB SF_OP3,MLIB	Sccanf option
SETLIB TLTB,MLIB	Sccanf option

## STDIO.H CONTENTS

```
IFDEF main
EXT $LM
EX SPC SET ccospace
TOT SZ SET 0
PUBLIC EX SPC,TOT_SZ
#endif bufsz1
$SIZ1 SET bufsz1
#endif
#endif bufsz2
$SIZ2 SET bufsz2
#endif
#endif bufsz3
$SIZ3 SET bufsz3
#endif
IRP X,<1,2,3,4,5,6>
IFNDEF $SIZ&X
$SIZ&X SET 256
ENDIF
$TAG&X SET -(X LE nfiles)
$OFF&X SET TOT SZ
TOT SZ SET $TAG&X*(36+$SIZ&X)+TOT_SZ
PUBLIC $TAG&X,$SIZ&X,$OFF&X
ENDIF
ENDIF
#endif lib1
LIB <lib1>
#endif
#endif lib2
LIB <lib2>
#endif
#endif lib3
LIB <lib3>
#endif
#endif mathlib
MLIB SET 1
#endif
USELIB ULIB,CLIBU
USELIB TLIB,CLIBT
USELIB XLIB,CLIBX
USELIB MLIB,CLIBM
USELIB main,CLIB
IFI
.PRINTX * STDIO.H - Ver 3.2 *
ENDIF
#endifasm
```

If main program, then insert hook to link in CCMAIN.REL.  
Set up extra space under BDOS  
Iterate to compute total fcb size  
Set up default file buffer sizes.  
Unix I/O should use 512 default.  
C/80 default is 256.

Compute file buffer+file control block sizes and grand total.

Set up tag for which ones used.

Invoke user library macros.

Usage is like  
#define lib1 V1,V2,V3  
entered prior to reading in STDIO.H.

Invoke user math library switch.

Unix library  
Transcendental function library  
Main library extension  
Math library  
Main library  
Print message on pass 1 of M80  
to tell user we're here and the C/80 compiler was used.

**== STANDARD LIBRARY CALLS ==**

Page	Cast	Function	Description
1	void	exit()	Abort exit, no flush or close.
1	void	<u>a</u> exit()	Abort exit.
2	int	abort(code)	Abort exit to DOT
4	int	abs(x)	Integer absolute value.
5	int	access(name,mode)	Access file boolean.
7	float	acos(x)	Arc Cosine.
8	float	acot(x)	Arc Cotangent.
9	float	acsc(x)	Arc Cosecant.
10	float	asec(x)	Arc Secant.
11	float	asin(x)	Arc Sine.
12	float	atan(x)	Arc tangent.
13	float	atan2(x,y)	Arc tangent of y/x.
14	float	atof(s)	Ascii to float.
15	int	atoi(s)	Ascii to integer.
16	long	atol(s)	Ascii to long.
17	int	bdos(code,DE)	BDOS interface.
19	int	binary(x,v,n)	Binary search.
23	int	bios(code,BC)	BIOS interface.
24	int	brk(address)	Program break.
26	char*	bsearch(key,base,n,w,cmp)	
62	void	C_cmln()	Move C command line.
28	char*	<u>c</u> alloc(m,s)	Alloc with null fill.
34	char*	Cbuf	Console buffer pointer.
34	char*	CC CCP	Stack pointer for caller of CCHAIN.
30	int	ccBDOS(DE,BC)	BDOS interface, low level.
32	int	ccBIOS(DE,BC,f)	BIOS interface, low level.
33	void	ccmain()	Main startup code.
35	char	CCTICK	Loop variable, terminal input.
36	float	ceil(f)	Ceiling, round up next integer.
34	void	Cexit()	Flush and close re-direction.
103	int	cfree(a)	Free address a from malloc().
38	unsigned	cfs(file)	Compute file size.
39	void	chain(tpa,program)	Chain to load address and run.
41	void	chdir(s)	Change default drive and user.
42	int	chmod(name,mode)	Change file attributes.
44	void	CHmode(n)	Change console mode.
46	int	chown(nm,own,gr)	Change file owner and group.
47	int	clearerr(fp)	Clear error on stream fp.
48	int	close(fd)	Close file primitive.
21	char	Cmode	Console echo=0, line=1, noecho=2.
49	int	cmpf(n,m)	Compare floats n,m.
49	int	cmpi(n,m)	Compare integers n,m.
49	int	cmpl(n,m)	Compare longs n,m.
49	int	cmps(n,m)	Compare strings n,m.
51	int	comp(str1,str2)	Compare, see instr().
52	int	con(c)	Single char console output.
52	int	coutout(cl,c2,...)	Multiple char console output.
53	int	Copen()	< & > redirection file opener.
54	char*	core(n)	Alloc memory from heap.
55	float	cos(x)	Cosine.

**— S T A N D A R D   L I B R A R Y   C A L L S —**

Page	Cast	Function	Description
56	float	cosh(x)	Hyperbolic cosine.
57	int	creat(name,pmode)	Ascii create file.
57	int	creata(name,pmode)	Same as creat().
57	int	creatb(name,pmode)	Binary create file.
58	char*	Ct1B	'B vector address data.
58	void	Ct1B()	'B processor, language C rules.
60	int	CtrlC()	Ctrl-C and Ctrl-B check.
61	char*	cvupper(s)	Convert string to upper case.
63	char*	decodF(fcb)	Decode file control block.
64	float	deg(x)	Conversion to degrees.
65	void	dsort(table)	Distribution sort.
67	void	dsort16(table)	Distribution sort, 16-bit numbers.
69	char*	edata,end,etext	System locations.
70	int	errno	Error return variable.
71	void	error(s1,s2,...)	Print strings to console.
72	void	exit()	Standard C exit. [29]
73	float	exp(x)	Power base e = 2.718 exponent x.
77	float	fabs(value)	Float Absolute value.
78	int	fclose(fp)	Flush and close stream.
82	int	fflush(fp)	Flush stream buffer to disk.
83	char*	fgets(s,n,fp)	Get a line from a stream.
84	int	fileno(fp)	Convert stream to descriptor.
85	char*	fillchr(dest,c,n)	Fill buffer with byte c.
86	int	fin	File descriptor for stream stdin.
87	int	fin()	Function to return fin.
88	char*	findFirst(e,f)	Find file f, extent e.
89	char*	findNEXT()	Find next after findFIRST().
90	void	fixfile(f,p)	Fix file f with pattern p.
91	float	floor(f)	Round down to integer.
92	float	fmod(x,y)	Remainder of x/y.
93	char*	fnb(s)	Find next blank or end of string.
94	FILE*	fopen(file,mode)	Standard C/80 file open.
96	FILE*	fopenT(file,mode)	Standard Unix file open.
96	FILE*	fopena(name,mode)	ASCII standard file open.
98	FILE*	fopenb(name,mode)	BINARY standard file open.
86	int	fout	File descriptor, stream stdout.
86	int	fout()	Function to return fout.
100	void	fprintf(fp,control,arg1,arg2,...)	
101	int	fputs(s,fp)	Print string s to stream fp.
102	int	fread(buff,s,n,fp)	Read n packets of size s.
103	int	free(a)	Free address a from malloc().
104	FILE*	freopa(nn,mode,fp)	Same as freopen().
105	FILE*	freopb(nn,mode,fp)	BINARY reopen file nn in new mode.
104	FILE*	freopen(nn,mode,fp)	ASCII reopen file nn in new mode.
107	float	frexp(x,ptr)	Break x into mantissa and exponent.
108	int	fscanf(fp,control,arg1,arg2,...)	
110	int	fseek(fp,off,pos)	Position file, long offset.
112	int	ftell(fp)	File position report, C/80.
113	long	ftell(fp)	File position, Unix.
112	int	ftellr(fp)	File position report, C/80.

**— STANDARD LIBRARY CALLS —**

Page	Cast	Function	Description
114	void	ftoa(how,pr,f,s)	Float to ASCII conversions.
115	int	fwrite(buff,s,n,fp)	Write n packets size s to stream.
21	char	GC BIN	Binary flag byte.
116	int	getatt(name)	CP/M coded attribute.
118	int	getbyte()	Get a char, no echo.
119	int	getc(fp)	Get a stream char ASCII.
120	int	getcbinary(fp)	Get a stream char BINARY.
121	int	getchar()	Get a stdin char.
122	int	getddir(s,p,n)	Get directory file names & sizes.
126	unsigned	getdfree(driv,dfp)	Get disk free space & info.
129	long	getl(fp)	Get stream long integer BINARY.
130	int	getline(s,n)	Read line, include newline.
131	int	getln(s,n)	Read line, drop newline.
133	char*	getpass(q)	Read in password noecho, device.
134	int	getpid()	Get process ID. Stub.
135	char*	gets(s)	Read a line, drop newline.
136	unsigned	getw(fp)	Read stream 16-bit word BINARY.
137	void	heaps(base,n,cmp)	Heap sort.
139	char*	highmem()	High memory address return.
140	float	Horner(x,p,n)	Horner's polynomial evaluation.
141	char*	index(str,c)	Report position of c in str.
142	int	inport(port)	Fetch byte from CPU port.
143	void	insert(tail)	String insert to CP/M buffers.
144	int	instr(n,str1,str2)	Find substring match position.
145	char	I0bin[MAXCHN]	Binary flag '\0' or 'b'.
145	char*	I0buf[MAXCHN]	Addresses of the file buffers.
145	char	I0ch[MAXCHN]	Channel numbers (-1 means empty).
145	char*	I0dev	Device string 'con:rdr:pun:lst:'
145	int	I0end[MAXCHN]	Saves EOF record number for seek.
145	char*	I0fcb[MAXCHN]	File control block addresses.
145	int	I0ind[MAXCHN]	Offset to next file buffer char.
145	char	I0mode[MAXCHN]	Mode byte 'r', 'w', 'u'.
145	int	I0nchx[MAXCHN]	Amount read by ffb() into buffer.
145	char	I0pchan[4]	Device numbers 252,253,254,255.
145	char	I0peof[4]	Device EOF chars 26,26,26,12.
145	char	I0pread[4]	CP/M read functions 1,3,0,0.
145	char	I0pwrite[4]	CP/M write functions 2,0,4,5.
145	char	I0rw[MAXCHN]	0 or 1 for last read or write.
145	int	I0sect[MAXCHN]	Sector count in 128-byte hunks.
145	int	I0size[MAXCHN]	Buffer size (256 default).
145	int	I0tmp	Storage, work variable.
150	int	isalnum(c)	Tests c for alphabetic or numeric.
150	int	isalpha(c)	Tests c for alphabetic.
150	int	isascii(c)	Tests c for range 0 to 127 decimal.
148	int	isatty(fd)	Test for console device.
150	int	iscntrl(c)	Test 0-31 decimal.
150	int	isdigit(c)	Tests for a valid digit, 0-9.
149	int	isgraph(c)	Test for graphics char.
150	int	islower(c)	-1 for 'a' <= c <= 'z', else 0.
150	int	isprint(c)	Test for a printable character.

**— S T A N D A R D   L I B R A R Y   C A L L S —**

Page	Cast	Function	Description
150	int	ispunct(c)	Test not ctrl nor alphanumeric.
150	int	isspace(c)	Test blank, tab, CR, LF.
150	int	isupper(c)	-1 for uppercase c, else 0.
150	int	isxdigit(c)	Test for hex digit, 0-9,A-F.
152	char*	itoa(x,s)	Integer to ASCII.
153	int	keystat()	Key and console buffer status.
154	long	labs(value)	Long Integer Absolute value.
155	float	ldexp(x,n)	Load new exponent into x.
156	float	ln(x)	Log base e, macro in math.h.
156	float	log(x)	Log base e, same as ln(x).
157	float	log10(x)	Log base 10, calculator log(x).
158	int	logged()	Return currently logged disk.
159	unsigned	loginv()	Return 16-bit disk log vector.
237	int	longjmp(env,val)	Software far goto, see setjmp.
160	char*	lownmem()	Low memory address.
161	char*	lsearch(key,base,n,w,cmp)	Linear search & insert.
163	char*	ltoa(x,s)	Long to ASCII.
164	void	makeFCB(fcb,name)	Make file control block from name.
165	char*	malloc(m)	Memory alloc.
65	struct	mathsort	Structure used in DSORT,DSORT16.
167	int	max(x,y)	Integer max.
37	int	MAXCHN[]	1+n, n = maximum number of files.
168	unsigned	memsize()	Amount of available heap memory.
168	int	midstr(s1,s2)	Same as instr(), but 2 args.
171	int	min(x,y)	Integer minimum.
172	char*	mkttemp(tmp)	Temporary file name generator.
173	float	modf(x,nptra)	Split float to mantissa & exp.
174	char*	moveMEM(dest,s,n)	Move n bytes no-ripple from s.
175	char*	movmem(s,dest,n)	Same as moveMEM, reverse args.
252	struct	mstr	MBASIC strings, used in ssort3().
177	int	numsort(n,table,s)	16-bit numerical sort.
179	int	open(name,access)	ASCII open, descriptor level.
179	int	opena(name,access)	ASCII open, descriptor level.
181	int	openb(name,access)	BINARY open, descriptor level.
183	int	outport(port,val)	Print byte to port.
34	char*	p_fin	Redirection input filename pointer.
34	char*	p_fout	Redirection output filename pointer.
21	char	PC BIN	Binary putc() flag.
184	char	peekb(addr)	Fetch byte at address.
184	long	peekl(addr)	Fetch long integer at address.
184	unsigned	peekw(addr)	Fetch unsigned word at address.
185	int	perror(s)	Print last system error.
187	void	pokeb(addr,x)	Deposit byte at address.
187	void	pokel(addr,x)	Deposit long integer at address.
187	void	pokew(addr,x)	Deposit word at address.
188	float	pow(x,y)	Standard power x to the y.
189	float	pow10(y)	Standard power 10 to the y.
190	int	printf(control,arg1,arg2,...)	
195	int	putc(x,fp)	Print byte to stream ASCII.
197	int	putcbinary(x,fp)	Print byte to stream BINARY.

**— S T A N D A R D   L I B R A R Y   C A L L S —**

Page	Cast	Function	Description
199	int	putchar()	Print byte to stdout ASCII.
200	long	putl(x,fp)	Print BINARY Long to stream.
201	int	puts(s)	Print string with newline.
202	unsigned	putw(n,fp)	Print BINARY word to stream.
203	int	qsort(base,n,s,CMP)	Quicksort, Unix standard.
205	void	quick(lo,hi,base,CMP)	Quicksort, alternate.
207	float	radian(x)	Radian value of argument x.
208	unsigned	rand()	Returns next random number.
208	unsigned	rand1()	Special service routine, rand.
209	int	rddISK(track,record,	drive,buffer) Direct-disk read.
294	unsigned	read(fp,buf,n)	C/80 read binary, 128 records.
212	int	readT(fd,buffer,n)	ASCII Unix read n bytes to buffer.
212	int	reada(fd,buffer,n)	ASCII read n bytes to buffer.
213	int	readb(fd,buffer,n)	BINARY read n bytes to buffer.
215	char*	realloc(ptr,s)	Re-allocate from malloc() call.
217	int	rename(old,new)	Rename a disk file.
218	void	reset()	Reset all drives.
219	char*	reverse(str)	Reverse string in place.
220	void	rewind(fp)	Position stream to start.
222	char*	rindex(str,c)	Reverse search string for char c.
223	void	run(cmd)	Run COM file with args.
224	int	sbinary(x,v,n)	Binary search string table.
225	char*	sbrk(n)	Basic heap allocate.
227	int	scanf(control,&arg1,&arg2,...)	
231	int	seek(fp,off,type)	Integer seek().
233	int	seekend(fd)	Seek to end of file.
234	void	seldisk(x)	Select disk by number.
235	int	setatt(name,code)	Set file attributes.
236	int	setbuf(fp,buffer)	Change file buffer location.
237	int	setjmp(env)	Set up far GOTO. See longjmp.
240	unsigned	sfree(n)	Free up space assigned by sbrk().
241	void	shells(p,n,CMP)	Shell-Metzner sort.
243	int	signal(sig,f)	Unix signal (interrupt).
244	float	sin(x)	Sine.
245	float	sinh(x)	Hyperbolic sine.
246	char*	sob(s)	Skip over blanks.
247	int	sprintf(s,control,arg1,arg2,...);	
248	float	sqrt(x)	Square root.
208	void	srand(seed)	Seeds the random number generator.
249	int	sscanf(s,control,arg1,arg2,...)	
251	void	ssort2(n,table,len)	Shell sort fixed length strings.
252	void	ssort3(size,&table)	Shell sort MBASIC string table.
253	#define	stderr (FILE *)252	
253	#define	stdin (FILE *)fin ()	
253	#define	stdout (FILE *)fout ()	
254	char*	strcat(str1,str2)	String concatenate.
255	char*	strchr(s,c)	Find c in string s.
256	int	strcmp(str1,str2)	Compare strings.
257	char*	strcpy(str1,str2)	Copy strings.
259	int	strcspn(s,set)	Find complement span from set.

— STANDARD LIBRARY CALLS —

Page	Cast	Function	Description
260	int	strlen(str)	String length.
261	char*	strncat(s1,s2,n)	Concatenate strings.
262	int	strcmp(s1,s2,n)	Compare strings.
263	char*	strncpy(s1,s2,n)	Copy strings /w NULL FILL.
264	char*	struprbrk(s,set)	Pointer span from set.
265	int	strpos(s,c)	Offset char c in string s.
266	char*	strrchr(s,c)	Pointer rev char c in string s.
267	char*	strrprbrk(s,set)	Pointer rev span from set.
268	int	strrpos(s,c)	Offset rev char c in string s.
269	int	strspn(s,set)	Offset span from set.
270	int	swab(s,d,n)	Swap words in memory.
271	int	system(s)	System call by \$\$.SUB.
272	float	tan(x)	Tangent.
274	float	tanh(x)	Hyperbolic tangent
275	void	timer(n)	H89/Z100 timer.
276	int	toascii(x)	Convert to ASCII range 0-127.
277	int	toint(x)	Convert hex byte to integer.
278	int	tokens(table,tail)	Command line decoder. [29]
280	int	tolower(x)	Convert to lower case.
33	EQU	TOT SZ	Total size of FCB & file buffers.
281	int	toupper(x)	Convert to upper case.
282	char*	ttyname(fd)	Name of console device.
82	int	uflush(fp)	Unix stream flush. See fflush.
284	int	ungetc(x,fp)	Push back char on stream.
285	EQU	unixio	All unix stub functions.
289	int	unlink(filename)	Erase directory entry.
290	char*	utoa(n,s)	Unsigned to ASCII base 10.
290	char*	utoab(n,s)	Unsigned to ASCII base 2.
290	char*	utoab(o,s)	Unsigned to ASCII base 8.
290	char*	utoax(n,s)	Unsigned to ASCII base 16.
291	void	waitz(n)	Time-out, H89/Z100.
292	int	wrDISK(track,record,drive,buffer)	Direct-disk write.
293	unsigned	write (fp,buf,n)	C/80 standard binary 128-write.
295	int	write(fd,buffer,n)	ASCII Unix write n bytes.
295	int	writea(fd,buffer,n)	ASCII write n bytes.
296	int	writeb(fd,buffer,n)	BINARY write n bytes.

== U N I X S Y S T E M C A L L S ==

Page	Cast	Function	Stub Return
285	int	acct(file)	Return 0.
285	unsigned	alarm(seconds)	Return 0.
285	char*	cuserid(s)	Return (char *)0.
285	int	dup(old)	Return -1 for error.
285	int	dup2(old,new)	Return -1 for error.
285	int	fork()	Return -1 for error.
285	int	fstat(fd,buf)	Return -1 for error.
285	void	ftime(tp)	Void return.
285	int	getpid()	Return 0.
285	int	getuid()	Return 0.
285	int	getgid()	Return 0.
286	int	geteuid()	Return 0.
286	int	getegid()	Return 0.
286	int	ioctl(fd,request,argp)	Return -1 for error.
286	int	getpgrp()	Return 0.
286	int	getppid()	Return 0.
286	int	kill(pid,sig)	Return -1 for error.
286	int	link(path1,path2)	Return -1 for error.
286	int	mknod(path,mode,dev)	Return -1 for error.
286	int	mount(spec,dir,rwflag)	Return -1 for error.
286	int	nice(incr)	Return 0.
286	void	pause()	Void return.
286	int	pclose(fp)	Return -1 for error.
286	int	pipe(fd)	Return -1 for error.
286	FILE*	popen(command,type)	Return 0.
286	void	profil(buff,bufsize,offset, scale)	Void return.
286	int	ptrace(request,pid,addr,data)	Return -1 for error.
286	int	putpwent(p,fp)	Return -1 for error.
286	int	setgid(gid)	Return -1 for error.
287	int	setpgrp()	Return 0.
287	int	setuid(uid)	Return -1 for error.
287	int	stat(name,buf)	Return -1 for error.
287	void	sync()	Void return.
287	long	time(tloc)	Returns (long)0
287	long	times(buffer)	Return -1 for error.
287	int	stime(tp)	Return -1 for error.
287	int	umask(cmask)	Return 0.
287	int	umount(spec)	Return -1 for error.
287	int	uname(name)	Return 0.
287	char*	userid(s)	Return (char *)0
287	int	ustat(dev,buf)	Return -1 for error.
287	int	utime(path,times)	Return -1 for error.
287	int	wait(stat_loc)	Return -1 for error.

**== U N I X M I S C E L L A N Y ==**

Page	Cast	Function	Stub Return
288	struct	group *getgrent()	Return 0.
288	struct	group *getgrgid(gid)	Return 0.
288	struct	group *getgrnam(name)	Return 0.
288	int	setgrent()	Return 0.
288	int	endgrent()	Return 0.
288	char*	getlogin()	Return (char *)0.
288	int	getpw(uid,buf)	Return -1 for error.
288	struct	passwd *getpwuid(uid)	Return 0.
288	struct	passwd *getpwent()	Return 0.
288	struct	passwd *getpwnam(name)	Return 0.
288	int	setpwent()	Return 0.
288	int	endpwent()	Return 0.
288	int	monitor(lowpc,highpc,buffer,bufsize,nfunc)	Return 0.
288	int	sleep(seconds)	Return 0.

# L I B R A R Y   I N D E X

"CON:", "LST:", "PUN:", "RDR:" ... 180, 182, 212, 282  
"r" mode ... 94, 95, 98, 104, 105  
"r+", "w+", "a", "+" file modes ... 94  
"rb" binary file mode ... 94  
"u" update file mode ... 94, 95  
"ub" update binary file mode ... 94  
"w" file mode ... 94, 95, 98, 104, 105  
"wb" write binary file mode ... 94  
\$DIR ... 5, 6, 42, 43, 116, 235  
\$END, \$END## ... 25, 34, 226  
\$END##+TOT SZ## ... 25, 226  
\$LM ... 34  
\$R/0, \$R/W, \$SYS ... 5, 6, 42, 43, 116, 235  
\$SIZ1, \$SIZ6 ... 146  
\$STRCV ... 290  
%% ... 190, 227, 284  
%b printf binary output ... 35, 194  
&-operator ... 75  
'con:rdr:pun:lst:' ... 145  
(\*cmp)() ... 26, 27, 137, 161, 162, 203, 205, 241  
(fscanf(fp, "%d", &d)) return value kludge ... 108  
(int)x.c[3]) float exponent access ... 107, 155  
+INF ... 8, 9, 10, 12, 272  
-128L ... 111  
-32767 ... 15, 163  
-INF ... 8, 9, 10, 12, 56, 156, 157, 188, 245, 272, 274  
-PI ... 9, 11, 12, 13  
-R-XR-XR-X ... 42  
-RWXRWXRWX ... 42  
04000 and similar octal access modes ... 42, 43  
0330 (H89 port) ... 142, 183  
0x5c, 0x6c, 0x80 default buffers ... 62, 63, 88, 223  
10mhz ... 170  
128-byte ... 38, 145  
128I ... 110  
136-character ... 35  
16-bit ... 15, 49, 67, 136, 142, 159, 177, 178, 187, 202, 208, 290  
16777216 ... 111, 113  
252, 253, 254, 255 device handles ... 78, 95, 145, 211, 294  
256\*\$IZBLOCK ... 166  
256-byte ... 112  
2n+512 distribution sort ... 177  
32-bit ... 14, 16, 17, 23, 49, 68, 100, 114, 129, 152, 163, 187, 200, 247  
32767 signed int limit ... 15, 16, 54, 111, 152, 163, 178, 225, 226, 260  
36-byte ... 18, 31, 164  
4-byte ... 103, 165  
4096 ... 115, 236  
4mhz clock speed ... 45, 178  
65535 unsigned limit ... 208, 240  
65536\*256 ... 111, 113  
68000 CPU ... 129, 136, 139, 170, 200, 202, 270  
8-bit ... 142, 187

# L I B R A R Y   I N D E X

80186,80286 CPU ... 270  
8080 CPU ... 15,68,139,152,174,178,270  
8086 CPU ... 139,176,270  
8088 CPU ... 176,270  
8250 UART ... 142,183  
8253 CTC ... 142,183,291  
<CTYPE.H> ... 150  
<GRP.H> ... 288  
<MATH.H> ... 7  
<PWD.H> ... 288  
<SYS/STAT.H> ... 285,287  
<SYS/TIMEB.H> ... 285,287  
<SYS/TYPES.H> ... 285,287  
exit() ... 1  
\_exit() ... 1,34,60  
abort(code) ... 2  
abs(x) ... 4  
access(name,mode) ... 5  
acct(file) ... 285  
accuracy ... 273,291  
acos(x) ... 7  
acot(x) ... 8  
acsc(x) ... 9  
alarm(seconds) ... 285  
algorithm ... 177,252  
analog-to-digital ... 142,183  
approximation ... 157,244,273  
archive ... 116,235  
asec(x) ... 10  
asin(x) ... 7,11  
atan(x) ... 12  
atan2(x,y) ... 13  
atof(s) ... 7,14  
atoi(s) ... 4,15  
atol(s) ... 16  
attribute ... 6,43,116  
auto-repeat ... 22  
auto-switching ... 174  
Banahan & Rutter ... 25,204,226,243,273,285  
BASIC Language ... 17,23,30,32,51,144,228,230,285  
bdos(code,de) ... 17  
BDS-C ... 17,23,175  
bell ... 199  
benchmark ... 85,275  
Bilofsky, Walt ... 146,232  
binary(x,v,n) ... 19  
bios(code,bc) ... 23,209,292  
bitmap ... 234  
blank-filled ... 164  
b1m ... 127  
boot+0x5c,boot+0x6c ... 143  
boot+0x80 ... 88,89

# L I B R A R Y   I N D E X

Bourne ... 141,222  
brk(address) ... 24,103,166,168,216,226  
bsearch(key,base,n,w,cmp) ... 26,162  
bsh ... 127  
buffered ... 22,35,79,82,94,142  
bufsz1,bufsz2,bufsz3 ... 146,220  
byte-sex ... 270  
C&H (Cheney & Hart) ... 12,157,188,244,248  
C cmln() ... 34,62  
calloc(m,s) ... 28,103  
Cbuf ... 34,59,143,153  
CC CCP ... 34  
ccBDO\$ (DE,BC) ... 30,89,128,158  
ccBIOS(de,bc,f) ... 32,128,133  
ccMAIN() ... 33,34  
CCP ... 62,143,223,271,279  
CCTICK ... 22,35,45  
ceil(f) ... 36  
CEXIT.CC ... 34  
Cexit () ... 34,37,53,72,78,279  
cfreeTa) ... 29,103  
cfs(file) ... 38  
chain(tpa,program) ... 39,143,271,279,  
chdir(s) ... 41  
checksum ... 127,128  
Cheney ... 12,244,273  
chmod(name,mode) ... 42  
Chnode(n) ... 44,121,182  
chown(name,owner,group) ... 46  
CI-86 ... 175  
clearerr(fp) ... 47  
clock ... 183,208,275,291  
close(fd) ... 48,57,179,181  
CMCLN.CC ... 34  
Cmode ... 21,22,34,35,45,59,60,118  
cmp(key,tbl),cmp(p,q) ... 26,161,242  
cmpf(n,m),cmpi(n,m),cmpl(n,m),cmps(n,m) ... 49,50,204  
Cody ... 244  
coefficients ... 244  
comp(str1,str2) ... 51  
con(c) ... 52  
concurrency ... 40  
conflush() ... 153  
conout(c1,c2,...) ... 52  
conversions ... 227,228,244  
COPEN.CC ... 34  
Copen () ... 34,53,279  
core( $\bar{n}$ ) ... 54,168,240,257,258  
cos(u)/sin(u) ... 273  
cos(x) ... 55  
cosecant ... 9  
cosh(x) ... 56

# L I B R A R Y   I N D E X

cosine ... 7,55,56  
cotangent ... 8  
CP/M-68k ... 17,23,129,136,139,200,202,209,292  
CP/M-80 ... 6,27,76,111,113,209,216,292  
CPMEOF ... 220  
creat(name,pmode) ... 57,79  
creata(name,pmode) ... 57  
creatb(name,pmode) ... 57  
cross-check ... 209  
CtlB,CtlB ... 34,58  
Ctlb () ... 34,45,58,59,60  
CtlCRT() ... 35,58,59,60  
ctrl-b ... 35,45,58,59,60,121,214  
ctrl-c ... 35,45,60,238  
ctrl-l ... 78  
ctrl-p ... 45  
ctrl-s ... 45  
ctrl-z ... 21,35,,83,120,121,123,135,195,201,211,232,233,294  
CTYPE.H ... 151,277  
cvupper(s) ... 61,133,141,222,224,256,262,281  
DDT ... 2,3,143  
debugger ... 2,3  
decodF(fcb) ... 63,88,89,164  
decodtokens(table) ... 278  
deg(x) ... 64,244,273  
degree ... 55,64,140,207,244  
dfreep->avail,dfreep->bposc,dfreep->spg,dfreep->total ... 128  
direct-disk ... 209,292  
dma ... 17,18,23,30,31,32  
Donald Knuth ... 65,66,177  
double-# ... 25,226  
dpbp->dsm,dphp->alvp,dphp->hdppb ... 127,128  
Dr.Dobbs Journal ... 209  
DRI ... 17,23  
DRIVER.PRE ... 39  
drm,dscr[3],dsm ... 127  
dskreset(drive) ... 218  
dsort(table) ... 65  
dsort16(table) ... 67,162  
dup(old) ... 285  
dup2(old,new) ... 285  
EBCDIC ... 276  
ECHO.C ... 201  
edata ... 25,69,226  
EDOM ... 7,10,11,13,156,157,188,248,272  
end-END.CC ... 34,69,146  
end-of-file ... 297  
endgrent() ... 288  
endpwent() ... 288  
env[1][1] ... 71,237,239  
environment ... 237,239

## LIBRARY INDEX

EOS ... 230  
eprom ... 71  
ERANGE ... 12,56,73,188,245,272,274  
Eratosthenes, Sieve of ... 85  
errno ... 70,91,107,155,173,185,207,244,272  
error(str1,str2,...) ... 71  
etext,ETEXT.C ... 25,69,226  
EX SPC ... 34  
exit() ... 34,72  
exam ... 127  
exp(x) ... 56,73,245,274  
expand(&argc,&argv) ... 74  
EXPLARGE ... 56,245,274  
exponent ... 14,73,155,173,188,189  
extent ... 194  
fabs(value) ... 77,273  
fcb[36] ... 164,175  
fclose(fp) ... 42,78,84,96,102,160,172,195,231,257,276,284,295  
fd=fopen(fp) ... 212,213,295,296  
fdopen(fd,mode) ... 79  
FDOS ... 40  
feof() ... 80,102,119,121,129,136  
ferror(fp) ... 81  
fflush(fp) ... 48,82,145  
fgets(s,n,fp) ... 5,83,230,257,284  
fileno(fp) ... 28,80,84,95,104,119,148,168,175,196,282  
fileprint() ... 58  
fillchr(dest,c,n) ... 85,292  
fin,fout ... 34,86  
fin (),fout () ... 86,87,253  
findFirst() ... 76,88,89,217  
findnext() ... 76,88,89,217  
fixfile(filename,pattern) ... 90  
floor(f) ... 91  
fnod(x,y) ... 92  
fnb(s) ... 93,246  
fopen (file,mode) ... 94,95  
fopenTname,mode) ... 5,42,96,175,195,200,257,276,284,295  
fopena(name,mode) ... 96  
fopenb(name,mode) ... 98,213  
fork() ... 285  
fout () ... 86,87,253  
fprintf(fp,control,arg1,arg2,...) ... 100,212,213,231,253  
fputs(s,fp) ... 5,101,201  
fread(buff,s,n,fp) ... 102,211  
free(p), cfree(p) ... 103,165  
freespace(d) ... 125  
freopa(name,mode,fp) ... 79,104  
freopb(name,mode,fp) ... 104,105  
freopen(name,mode,fp) ... 104,106,118,279,284  
frexp(x,nptr) ... 107  
fscanf(fp,control,&arg1,&arg2,...) ... 108,109

# L I B R A R Y   I N D E X

fscanf return kludge ... 109  
fseek(fp,offset,position) ... 110,145,221,232,284  
fstat(fd,buf) ... 285  
ftell(fp),ftellr(fp) C/80 ... 112  
ftell(fp),ftellu(fp) Unix ... 113  
ftime(tp) ... 285  
ftoa(how,pr,f,s) ... 14,114  
fwrite(buff,s,n,fp) ... 115,294  
GC BIN ... 21,22,118  
getatt(name) ... 116  
getbyte() ... 22,35,118,153,214  
getc(fp) ... 98,110,119,145,211,220,276,283  
getcbinary(fp) ... 22,120,214,238  
getchar() ... 60,82,105,118,121,277  
getddir(s,p,nmax) ... 122,123  
getdfree(drive,dfp) ... 125,126,127  
getegid() ... 286  
geteuid() ... 286  
getgid() ... 46,285  
getl(fp) ... 129,200  
getline(s,n) ... 130,219  
getln(s,n) ... 131  
getlongs(fp) ... 129  
getmem() ... 103,165  
getmypassword() ... 133  
getpass(q) ... 133  
getpgid() ... 286  
getpid() ... 134,285  
getppid() ... 286  
getpw(uid,buf) ... 288  
gets(s) ... 75,93,135,246  
getuid() ... 46,285  
getw(fp) ... 136  
getwords(fp) ... 136  
graphics ... 52,142,149,183  
H89 ... 275,291  
handles ... 122,278  
Harbison & Steele ... 29,122,138,141,190,222  
Hart & Cheney ... 12,244,273  
heaps(base,n,cmp) ... 27,137,162  
heapsize ... 166  
heapsort ... 137  
Heath/Zenith ... 2,3  
highmem() ... 139,168  
Horner(x,p,n) ... 140  
Horner's method ... 140  
Hyperbolic ... 245,274  
i-node ... 42  
IBM ... 276  
index(str,c) ... 141  
initSystem() ... 71  
inport(port) ... 142

# L I B R A R Y   I N D E X

insert(tail) ... 143  
instr(a\$,b\$) ... 51  
instr(n,str1,str2) ... 144  
INTEL ... 85,129,136,142,183,200,202,270  
interrupt ... 183,243  
interrupt-driven ... 35  
I0bin[maxchn] ... 145,233  
I0buf[fd] ... 145,147,240  
I0BYTE ... 21  
I0ch[maxchn] ... 37,95,145,240  
I0ctl(fd,request,argp) ... 286  
I0dev ... 145  
I0end[maxchn] ... 145  
I0fcb[maxchn] ... 145,147,175,240  
I0ind[maxchn] ... 145,233  
I0mode[maxchn] ... 80,145  
I0nchx[maxchn] ... 145  
I0pchan[4] ... 145  
I0peof[4] ... 145  
I0pread[4] ... 145  
I0pwrit[4] ... 145  
I0rw[maxchn] ... 145  
I0sect[maxchn] ... 145,211,294  
I0size[maxchn] ... 145,147,236  
IOTABLE ... 145  
IOTABLE.CC ... 69,95,146,147,233  
I0tmp ... 145  
isalnum(c) ... 150,151  
isalpha(c) ... 150,151  
isascii(c) ... 150,151  
isatty(fd) ... 148  
iscntrl(c) ... 150,151  
isdigit(c) ... 150,151,230  
isgraph(c) ... 149  
islower(c) ... 150,151  
isprint(c) ... 149,150,151  
ispunct(c) ... 150,151  
isspace(c) ... 93,150,151,246  
isupper(c) ... 150,151  
isxdigit(c) ... 150,151  
itoa(number,s) ... 152,290  
jmpbuffer,jmp buf ... 71,237,238,239  
joysticks ... 142,183  
K&R (Kernighan & Ritchie) ... 103,114,120,132,150,166,192,194,197  
Kernighan ... 242  
keyboards ... 22  
keystat() ... 153  
kill(pid,sig) ... 286  
kludge (multi-arg) ... 109,182,194,230,250  
Knuth, Donald ... 65,177,251  
L80 ... 1,271  
labs(value) ... 4,85,95,105,112,154,210,217,232,289,293

## L I B R A R Y   I N D E X

Lattice C Compiler ... 175  
ldexp(x,n) ... 155  
left-justification ... 190  
LIB.COM ... 146  
link(path1,path2) ... 286  
ln(x) ... 156  
log(x) ... 17,18,30,31,156,157,234  
log10(x) ... 157  
logarithm ... 156,157  
logged() ... 125,158,218  
loginv() ... 159  
longjmp(env,v),setjmp(env) ... 58,71,237,238,239  
lownmem() ... 139,160  
LRU method ... 209,292  
lsearch(key,base,n,w,cmp) ... 27,161  
ltoa(x,s) ... 163  
M80 ... 25,112,226,271  
machine-dependent ... 142,183  
macros ... 150,151,277  
makefcb(fcb,file) ... 38,88,164  
malloc(m) ... 29,103,165,215  
mantissa ... 155  
math.h ... 4,156,167,171  
mathpack ... 14,114  
mathsort ... 65,66,67,68,177  
max(x,y) ... 167  
MAXCHN[] ... 37,95,145,236  
maxfiles ... 76  
maxsector ... 23,32  
maxtrack ... 23,32  
memsize() ... 139,168,225  
mice,mouse accessory ... 142,183  
Micropromo ... 276  
Microsoft ... 1,170,252,263  
midstr(str1,str2) ... 169  
min(x,y) ... 171  
mknode(path,mode,dev) ... 286  
mktemp(name) ... 172  
modem ... 142,183  
modf(x,nptra) ... 173  
monitor(lowpc,highpc,buffer,bufsize,nfunc) ... 208  
motorola ... 129,136,139,200,202  
mount(spec,dir,rwflag) ... 286  
moveMEM(dest,source,n) ... 174,254,258  
movmem(source,dest,n) ... 175  
multiple-argument ... 52,71,109,250  
mycmp(p,q) ... 138,203,242  
myctrlb() ... 58  
mypassword ... 133  
n=peekb(0x80) ... 62  
Newton Method ... 248  
nice(incr) ... 286

## L I B R A R Y   I N D E X

no-echo ... 17,30,35,60,118,214  
nomatch(n,f,t) ... 76  
non-interrupt ... 22,35  
non-portable ... 16,100,163,198,209,226,292  
non-ripple memory move ... 175  
numsort(n,table,scrap) ... 20,27,177  
obsolete Unix calls ... 86  
open(name,access) ... 48,179  
opena(name,access) Unix open ... 179,181  
openb(name,access) Binary open ... 181  
outport(port,value) ... 142,183  
p\_fin,p\_fout ... 34,105  
padding ... 190,193  
PASCAL Language ... 85  
pause() ... 286  
PC-DOS operating system ... 41  
PC BIN,GC BIN,Cmode ... 21,22  
pcTosef(fpT) ... 286  
peekb(addr) ... 184  
peekl(addr) ... 66,184  
peekw(addr) ... 26,38,67,184  
pens, light ... 142,183  
perror(s) ... 70,185  
pfFLOAT ... 7,36,55,64,73,77,91,107,140,155,173,185,207,228,244,272  
pfLONG ... 16,38,66,154,184,187,229  
PI ... 7,8,9,10,11,12,13,272,273  
pipe(fd) ... 286  
plotters ... 142,183  
pokeb(addr,x),pokel(addr,x),pokew(addr,x) ... 187  
polynomial ... 140  
portability ... 151,170,180,182,209,239  
pow(x,y) ... 73,188,248  
pow10(y) ... 189  
precision ... 114,190,191,192,193,228  
Prentice-Hall ... 138,242  
printf(control,arg1,arg2,...) ... 108,119,121,190,249  
prnt 1(),prnt 2 ... 192,193  
processforeignfiles() ... 22  
profil(buff,bufsize,offset,scale) ... 286  
pseudo-random ... 208  
ptrace(request,pid,addr,data) ... 286  
pump(buffer,file) ... 179,181  
pumpword(fp,x) ... 202  
push-back ... 283,284  
put2048(fp,buf) ... 293  
putc(x,fp) ... 52,80,95,101,115,195,276,277,294,295  
putcbinary(x,fp) ... 22,197  
putchar('7') ... 45  
putchar(x) ... 47,60,81,98,110,199,236  
putl(x,fp) ... 200  
putpwent(p,fp) ... 286  
puts(s) ... 152,153,201,290

## L I B R A R Y   I N D E X

putw(n,fp) ... 80,202  
qsort(base,n,s,cmp) ... 49,203  
quanta ... 95,211,294,295,297  
quick(lo,hi,base,cmp) ... 27,205,224,251  
quicksort ... 203,205  
rad(x) ... 55,207,244,273  
Raike ... 204,206  
rand(),rand1() ... 208  
random ... 5,18,31,82,95,263  
rdDISK() ... 209  
re-direction ... 148,199,253,278,284,295  
read(fp,buffer,count) C/80 read ... 211,294  
read(fd,buf,n) Unix read ... 28,84,102,212  
reada(fd,buffer,n) Unix read ... 102,212  
readb(fd,buffer,n) Binary read ... 213  
realloc(ptr,s) ... 215  
recursive ... 194,247  
rename(oldname,newname) ... 18,31,217  
report(fp) ... 112  
reset() ... 218,234  
reverse(str) ... 219  
rewind(fp) ... 5,220,221,294  
rewind(fp) code macro ... 221  
rindex(str,c) ... 222  
Ritchie ... 242  
robotics ... 71  
ROM ... 52,71,233  
RST instruction ... 3  
run(cmd) ... 223,272,279  
Rutter ... 25,204,226,243,285  
Sailor ... 51,144  
sbinary(x,v,n) ... 27,162,224  
sbrk(n) ... 160,225  
scanf(control,&arg1,&arg2,...) ... 227,228,230,246,284  
scientific ... 191  
scrap[10+256] ... 177  
secant ... 10  
SECSIZ ... 127,128  
sectors/track ... 127  
seek(fp,offset,type) C/80 standard ... 80,95,145,231,284  
seekend(fd) ... 97,99,180,182,233  
seldisk(x) ... 127,234  
setatt(name,code) ... 43,235  
setbuf(fp,buffer) ... 236  
setgid(gid) ... 46,287  
setgrent() ... 288  
setjmp(env) ... 58,71,237,239  
SETJMP.H ... 237,239  
setmem() ... 85  
setpgrp() ... 286  
setpwent() ... 288  
setuid(uid) ... 46,287

# L I B R A R Y   I N D E X

sfFLOAT ... 109,228,230,250,287  
sfLONG ... 109,229,230,250  
sfree(n) ... 29,95,103,146,147,166,178,240  
sgtty ... 286  
Shell-Metzner sort ... 251,252  
shells(p,n,cmp) ... 27,162,241  
SID ... 2,3  
Sieve of Eratosthenes ... 85,275  
signal() ... 243  
sin(x),sine ... 11,70,185,244,245,273  
sinh(x) ... 245  
SIZBLOCK ... 166  
sleep(seconds) ... 288  
sob(s) ... 75,93,246  
sorted ... 122,124,224,251,252  
SPG (sectors per group) ... 124,126,127  
sprintf(s,control,arg1,arg2,...) ... 100,114,152,247  
SPT (sectors per track) ... 127  
sqrt(x) ... 248  
srand(seed) ... 208  
sscanf(s,control,&arg1,&arg2,...) ... 249,250  
sscanf return kludge ... 250  
ssort2(size,table,reclen) ... 27,122,125,162,251  
ssort3(size,&table) ... 27,162,252  
stat(name,buf) ... 287  
stdout ... 279  
Steele & Harbison ... 29,138,141,190,222  
stime(tp) ... 287  
strcat(str1,str2) ... 254  
strchr(s,c) ... 141,255  
strcmp(str1,str2) ... 49,137,138,256  
strcpy(str1,str2) ... 90,257  
strcspn(s,set) ... 259  
strlen(str) ... 141,260  
strncat(str1,str2,n) ... 85,261  
strncmp(str1,str2,n) ... 262  
strncpy null fill warning ... 263  
strncpy(str1,str2,n) ... 62,85,263  
struprbrk(s,set) ... 264  
strpos(s,c) ... 265  
 strrchr(s,c) ... 222,266  
strrpbrk(s,set) ... 267  
strrpos(s,c) ... 268  
strspn(s,set) ... 269  
stub ... 97,243  
swab(s,d,n) ... 270  
switch(setjmp(env)) ... 71,238  
sync() ... 287  
system(s) ... 41,271  
tablets, graphics ... 142  
tangent,tan(x) ... 12,13,272,273,274  
tanh(x) ... 274

## LIBRARY INDEX

Taylor polynomial ... 273  
time(tloc) ... 287  
timer(n) ... 142,169,183,275  
times(buffer) ... 287  
toascii(x) ... 276  
toint(x) ... 277  
tokens(table,cmdtail) ... 75,76,143,278  
tolower(x) ... 280  
Toolworks, The Software ... 94,114,210,280,293  
TOT SZ ... 25,34,226  
toupper(x) ... 61,277,281  
TPA (transient program area) ... 39,40  
transcendental ... 14,70,114,186  
ttyname(fd) ... 282  
type-aheads ... 22,35,59  
typedef ... 239  
UART (universal async rec/trans) ... 142,183  
uflush() ... 82  
umask(cmask) ... 287  
umount(spec) ... 287  
uname(name) ... 287  
unbuff(fd) ... 240  
unbuffered ... 180,182  
ungetc(x,fp) ... 80,230,283,284  
union ... 107,187  
UNIXIO.H ... 285  
unix I/O and system functions ... 285  
unlink(filename) ... 160,175,217,289  
ustat(dev,buf),utimbuf,utime(path,times) ... 287  
utoa(n,s),utoab(n,s),utoao(n,s),utoax(n,s) ... 290  
wait(stat loc) ... 287  
waitz(n) ... 291  
Waite and Cody ... 244  
wakeports() ... 71  
whatdevice(fp) ... 282  
while(keystat()) ... 153  
Wiley, John & Sons ... 25,204,226,243,285  
Wizard C Compiler ... 175  
Wordstar (Micropac) ... 276  
wrDISK(trk,sec,dsk,buf) ... 292  
write (fp,buffer,count) C/80 standard ... 293,294  
write(fd,buffer,n) Unix write ... 48,84,95,179,181,293,294,295,297  
writea(fd,buffer,n) Unix write ... 115,295  
writeb(fd,buffer,n) Binary write ... 57,196,199,296  
write-protect ... 18,31,235  
Z100 ... 208,275,291  
Z29 ... 52  
Z80 CPU ... 142,174,176,178,183,270,291  
Z90/H89 Zenith Computer ... 183,209,292

## UNDERSTANDING LIBRARY DESCRIPTIONS

Each separate function is documented according to the following scheme:

- o Purpose
- o Function Header
- o Returns
- o Example
- o Notes

Function descriptions are listed alphabetically. It is normal to use just one page per function. Use the headline at the top of the page to locate a particular function.

**o Purpose.** A sentence or two which attempts to describe what the library function or variable is supposed to do or accomplish.

**o Function Header.** For each function in the library archive, we extract the function header and explain the nature and purpose of the variables in the argument list, plus the return value of the function.

1. The function name and the order of its arguments.
2. The function return value, e.g., long, int, float, char\* or void. Return values are 16 bits or 32 bits.
3. The data storage class of each argument. Each argument uses 16 bits (2 bytes) except for float and long, which use 32 bits (4 bytes). All pointer types use 16 bits.

**o Returns.** The function can return values in several ways, which are documented in this section of the function description:

1. The function returns a value in the processor registers, which is directly usable by language C, e.g.,  $i = f(x)$ .
2. Variables passed in the argument list of the function are pointers to data locations. The function may alter the contents of these data locations.
3. Global variables may be altered by a function, e.g., the error return variable **errno** in the transcendental function library.
4. Disk I/O can change the contents of file buffers, and the position of the file pointer, e.g., `getc()` and `fseek()`.

**o Example.** Where possible, an example or two is given that illustrates how to use the function. Examples tend to be test programs, e.g., a complete main program that illustrates the central ideas.

**o Notes.** Each function has implementation notes or special problems that are unique to its usage. Included here are cross-references to other functions and the limitations of usage.

## **UNDERSTANDING LIBRARY DESCRIPTIONS**

**Copyright (1985) by G.B.Gustafson and Viking C Systems  
All Rights Reserved**

## \_exit and a\_exit

### The \_exit Function

#### Purpose:

Abort exit. Does not flush buffers nor close the file control block to the disk directory.

#### Function Header:

```
#include <stdio.h>
VOID _exit()
This is a macro equivalent to a exit().
Stdio.h contains the macro definition.

VOID a_exit()
Actual function name in CLIB.REL.
See also CCMAIN.CC.
```

#### Returns:

Nothing. Exits to warm boot.

#### Example: Use of \_exit() to purge contents of re-direction files.

The command line is: A>main >foo.bar. The file foo.bar will have zero records.

```
#include <stdio.h>
main()
{
    printf("This message will not make it to disk\n");
    exit();
    printf("This line will never be printed\n");
}
```

#### Notes:

- o Standard abort exit. Calls BDOS function 0, but does not flush buffers or close files.
- o Due to problems with the Microsoft linker, the internal symbol used is A\_EXIT rather than \_EXIT.
- o The correct Unix symbol is exit(). Source code should use this symbol exclusively. The global symbol used by the link editor L80.COM is A\_EXIT.

## a b o r t

### The abort Function

#### Purpose:

Debugger support for C programs. Can be used with DDT, SID.

#### Function Header:

abort(code)	Abort exit to DDT
int code;	Code passed to machine register HL

#### Returns:

Caller	If DDT is not loaded
DDT	If DDT & C program loaded together

**Example:** Call sbrk() for more memory until the request fails, then break to DDT.

```
main(argc,argv) int argc; char **argv;
{
    char *sbrk();

    if(sbrk(10240) == (char *)-1) abort(1);
    if(sbrk(10240) == (char *)-1) abort(2);
    if(sbrk(10240) == (char *)-1) abort(3);
    if(sbrk(10240) == (char *)-1) abort(4);
    if(sbrk(10240) == (char *)-1) abort(5);
    if(sbrk(10240) == (char *)-1) abort(6);
    if(sbrk(10240) == (char *)-1) abort(7);
}
```

This program jumps to DDT the first time it fails to obtain another 10k of memory from sbrk(). To re-enter the C program from DDT, note the DDT stop address and do a GO command at that address plus 1 (e.g., -G0a31 if the break was at 0a30 hex).

For example, if the failure occurred at level 5, then DDT would intercept inside the abort() code with HL=5. You can step through the rest of the program, each time re-entering DDT with HL=6, HL=7, until finally the exit() is reached.

**WARNING:** This abort() is set up for a Heath/Zenith machine using CP/M 2.2 with interrupt 7 reserved for DDT. To get it to function under other CP/M versions or on other hardware, it may be necessary to re-assemble the source code.

## a b o r t

### Notes:

- o The value of the argument is in register HL before the call and in register BC after the call.

- o To use DDT, it must be invoked with the program:

A>DDT MYPROG.COM

- o Warning: Like all debugger code, do not leave it in place in the final program.

- o This implementation uses the standard Heath CP/M RST location for DDT/SID. It should be a NOP if DDT or SID is not loaded with the C program.

- o Both DDT and SID corrupt the interrupt vectors for RST 7. Object code with abort() calls is not end-user object code. Remove abort() calls by placing the source code `abort(){} with main()`.

- o If your system is different, then it MUST be changed for the hardware and re-compiled.

## The abs Function

### Purpose:

Computes the absolute value of an integer argument.

### Function Header:

```
int abs(x)
int x;           Integer to transform.
```

### Returns:

The positive integer obtained from x by removing the sign, i.e., returns x if x was already positive, else returns -x.

**Example:** Compute the absolute value of the difference between two numbers a and b.

```
#include <stdio.h>
main()
{
char s[129];
int a,b;

printf("Enter number a: ");
gets(s); a = atoi(s);
printf("Enter number b: ");
gets(s); b = atoi(s);
printf("abs(%d) - (%d) = %d\n",a,b,abs(a-b));
}
```

### Notes:

- o The abs() function used here is an honest function that works only on integers. It fails on long integers and floats.
- o For long integers, use labs().
- o For floats, use fabs().
- o Most libraries assume the abs() function is a macro defined in the STDOIO.H header file or in MATH.H. Such functions definitely have side effects. Beware when you port the code. Engineer against side effects when you write it for the first time.

## a c c e s s

### The access Function

#### Purpose:

Checks legal access to a file. Under CP/M, the check consists of a search for a matching directory entry, plus matching file attributes (\$R/O, \$R/W, \$DIR, \$SYS).

#### Function Header:

int access(name,mode)	Access boolean 0 or -1
char *name;	File name string pointer
int mode;	Unix mode integer, see below

#### Returns:

0	access allowed (file exists with attributes)
-1	access denied (file not found or bad attributes)

**Example:** Look up the file entered on the command line to see if it is in the disk directory and can be updated as a random file in read/write mode.

```
#include <unix.h>
#include <stdio.h>
help()
{
    puts("Usage: A>main filename"); exit();
}

main(argc,argv) int argc; char **argv;
{
FILE *fp,*fopen();
char buf[128];
int access();
    if(argc <= 1) help();
    if(access(argv[1],02)) puts("File not found or $R/O");
    fp = fopen(argv[1],"a+");
    if(fp == (FILE *)0) { puts("Bad open"); exit();}
    rewind(fp);
    fgets(buf,80,fp);
    fputs(buf,stderr);
}
```

## a c c e s s

### Notes:

- o The name is a pointer to a null-terminated file name. If the drive is missing, then it is assumed to be the currently logged drive.
- o The mode value is the Unix mode, which under CP/M causes a directory lookup to see if the protection bits match:

UNIX MODE	DESCRIPTION OF ACTION	CP/M ATTRIBUTE
00	check directory access	file exists
01	check exec access	\$DIR
02	check write access	\$R/W
04	check read access	file exists

- o CP/M-80 only checks to see if the file is in the default directory. No directory search path is performed.

### The acos Function

---

#### Purpose:

Compute the arc cosine of real float value x. The value x is assumed between -1 and 1.

#### Function Header:

float acos(x)	Arc Cosine.
float x;	Float value for computation.

#### Returns:

0	Range error, set errno = EDOM
angle	No error, answer in radians 0..PI

#### Example:

```
#define pfFLOAT 1
#include <math.h>
#include <stdio.h>
main(argc,argv) int argc; char **argv;
{
    char s[129];
    float x;
    extern int errno;
    float atof();
    float acos();
    errno = 0;
    printf("Enter a float: "); gets(s); x = atof(s);
    printf("acos(%f) = %f, errno = %d\n",x,acos(x),errno);
}
```

#### Notes:

- o The method used is  $\text{acos}(x) = \pi/2 - \text{asin}(x)$ .

## a c o t

### The acot Function

#### Purpose:

Compute the arc cotangent of real float value x. The value x is assumed between -INF and +INF.

#### Function Header:

```
float acot(x)          Arc Cotangent.  
float x;               Float value for computation.
```

#### Returns:

angle answer in radians 0 .. PI approximately.

#### Example:

```
#define pfFLOAT 1  
#include <math.h>  
#include <stdio.h>  
main(argc,argv) int argc; char **argv;  
{  
char s[129];  
float x;  
extern int errno;  
float atof();  
float acot();  
errno = 0;  
printf("Enter a float: "); gets(s); x = atof(s);  
printf("acot(%f) = %f, errno = %d\n",x,acot(x),errno);  
}
```

#### Notes:

- o The method used is  $\text{acot}(x) = \text{PI}/2 + \text{atan}(-x)$ .

### The acsc Function

---

**Purpose:**

Compute the arc cosecant of real float value x. The value x is assumed between -INF and +INF.

**Function Header:**

```
float acsc(x)          Arc Cosecant.  
float x;                Float value for computation.
```

**Returns:**

angle      Answer in radians -PI/2 .. PI/2 approx

**Example:**

```
#define piFFLOAT 1  
#include <math.h>  
#include <stdio.h>  
main(argc,argv) int argc; char **argv;  
{  
char s[129];  
float x;  
extern int errno;  
float atof();  
float acsc();  
  
errno = 0;  
printf("Enter a float: "); gets(s); x = atof(s);  
printf("acsc(%f) = %f, errno = %d\n",x,acsc(x),errno);  
}
```

**Notes:**

- o The method used is  $\text{acsc}(x) = \text{asec}(x/\sqrt{x^2-1})$ .

## a s e c

### The asec Function

#### Purpose:

Compute the arc secant of real float value x. The value x is assumed between -INF and +INF.

#### Function Header:

```
float asec(x)          Arc Secant.  
float x;                Float value for computation.
```

#### Returns:

0	Range error, set errno = EDOM.
angle	No error, answer in radians 0 .. PI.

#### Example:

```
#define pfFLOAT 1  
#include <math.h>  
#include <stdio.h>  
main(argc,argv) int argc; char **argv;  
{  
char s[129];  
float x;  
extern int errno;  
float atof();  
float asec();  
errno = 0;  
printf("Enter a float: "); gets(s); x = atof(s);  
printf("asec(%f) = %f, errno = %d\n",x,asec(x),errno);  
}
```

#### Notes:

- o The method used is:

$$\text{asec}(x) = \text{atan}(\sqrt{x*x - 1}) - (x \geq 0 ? 0 : \pi);$$

### The asin Function

---

#### Purpose:

Compute the arc sine of real float value x. The value x is assumed between -1 and 1.

#### Function Header:

float asin(x)	Arc Sine.
float x;	Float value for computation.

#### Returns:

0	Range error, set errno = EDOM.
angle	No error, answer in radians -PI/2 .. PI/2.

#### Example:

```
#define pFFLOAT 1
#include <math.h>
#include <stdio.h>
main(argc,argv) int argc; char **argv;
{
char s[129];
float x;
extern int errno;
float atof();
float asin();
    errno = 0;
    printf("Enter a float: "); gets(s); x = atof(s);
    printf("asin(%f) = %f, errno = %d\n",x,asin(x),errno);
}
```

#### Notes:

- o The method used is  $\text{asin}(x) = \text{atan}(x/\sqrt{1 - x^2})$ .

## The atan Function

### Purpose:

Compute the arc tangent of real float value x. The value x is assumed between -INF and +INF.

### Function Header:

```
float atan(x)
float x;           Float value in range -INF to +INF.
```

### Returns:

angle	Answer in radians -PI/2 .. PI/2 approx
0.0	ERANGE error

### Example:

```
#define pFFLOAT 1
#include <math.h>
#include <stdio.h>
main(argc,argv) int argc; char **argv;
{
char s[129];
float x;
extern int errno;
float atof();
float atan();
errno = 0;
printf("Enter a float: "); gets(s); x = atof(s);
printf("atan(%f) = %f, errno = %d\n",x,atan(x),errno);
}
```

### Notes:

- o The method used is rational approximation, Cheney & Hart.
- o Constants from C&H, 6.5.21, 6.5.22, Table 5097.

## a t a n 2

### The atan2 Function

---

#### Purpose:

Compute the arc tangent of real float value  $y/x$  to find the angle in radians between the x-axis and the ray through  $(0,0)$  and  $(x,y)$ .

#### Function Header:

```
float atan2(x,y)  
float x,y;           Float values for the point (x,y).
```

#### Returns:

0	Error $x=y=0$ , $\text{errno} = \text{EDOM}$
angle	Answer in radians $-\pi .. \pi$ approx
$\pi/2$	$x=0, y>0$
$-\pi/2$	$x=0, y<0$

#### Example:

```
#define pfFLOAT 1  
#include <math.h>  
#include <stdio.h>  
main(argc,argv) int argc; char **argv;  
{  
char s[129];  
float x,y;  
extern int errno;  
float atof();  
float atan2();  
errno = 0;  
printf("Enter float x: "); gets(s); x = atof(s);  
printf("Enter float y: "); gets(s); y = atof(s);  
printf("atan2(%f,%f) = %f, errno = %d\n",x,y,atan2(x,y),errno);  
}
```

#### Notes:

- o The function exists to handle the likely boundary cases. Users should study the return values carefully and include error checks in all source code that uses atan2().

## a t o f

### The atof Function

#### Purpose:

Converts a null-terminated string of decimal digits into an internal format 32-bit floating point number.

#### Function Header:

```
float atof(s)
char *s;           String of decimal digits.
```

#### Returns:

Floating point number, single-precision, 32-bits.

**Example:** Read a float from the console, print its value.

```
#define pfFLOAT 1
#include <stdio.h>
main()
{
    float f,atof();
    char s[129];

    printf("Enter a float: ");
    gets(s);
    f = atof(s);
    printf("float f = %f\n",f);
}
```

#### Notes:

- o The cast of atof() is (float). It must be declared before use. Failure to do so may crash your machine.
- o Floats are 32 bits (4 bytes). Storage of a float is documented in the C/80 Mathpack. See also the Transcendental function library for information about the floating point exponent.
- o The sources for ATOF and FTOA are part of the MathPack. Use of atof() invokes the float library.
- o Float atof() is used to decode strings in fixed or scientific formats. Strings like "84745475.237237" or "1.0e-16" are acceptable. Atof() is a Unix function.
- o Float allows decimal points, leading +-, exponent e or E, and a sign in the exponent. A floating point number cannot contain imbedded white space. Non-digits cause the decoding to stop. This includes white space or the end of the string.

## The atoi Function

---

### Purpose:

Converts a null-terminated string of decimal digits into a 16-bit internal format integer.

### Function Header:

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

String of decimal digits.

### Returns:

Integer in the range -32767 to 32767. The result is signed.

**Example:** Read an integer from the console, print its value.

```
#include <stdio.h>
main()
{
    int i;
    char s[129];

    printf("Enter a signed integer: ");
    gets(s);
    i = atoi(s);
    printf("i = %d\n",i);
}
```

### Notes:

- o The string may have leading white space and a leading sign next to the first decimal digit. Digits must be from the character set "0123456789". No hexadecimal digits are allowed. Leading zeros do not signify octal for this function - they are ignored.
- o Unsigned integers greater than 32767 will be treated as long integer data by the compiler, unless a suitable cast is imposed on the number.
- o The internal storage of an integer is two bytes (16 bits), stored in the usual 8080 reverse format. Integers typed at the terminal are in ascii decimal format, using the character set '0',..., '9'.
- o Non-digits cause the decoding to stop. This includes white space or the end of the string.

# a t o l

## The atol Function

### Purpose:

Converts a null-terminated string of decimal digits into an internal format 32-bit signed long integer.

### Function Header:

```
long atol(s)
char *s;           String of decimal digits.
```

### Returns:

Signed long integer.

**Example:** Read a long integer from the console, print its value.

```
#define pfLONG 1
#include <stdio.h>
main()
{
long i,atol();
char s[129];

printf("Enter a signed integer: ");
gets(s);
i = atol(s);
printf("long i = %ld\n",i);
}
```

### Notes:

- o The cast of `atol()` is `(long int)`. It must be declared before use. Failure to do so may crash your machine.
- o The string may have leading white space and a leading sign next to the first decimal digit. Digits must be from the character set "0123456789". No hexadecimal digits are allowed. Leading zeros do not signify octal for this function - they are ignored.
- o Unsigned integers greater than 32767 will be treated as long integer data by the compiler, unless a suitable cast is used.
- o Long integers are 32 bits (4 bytes). Storage of a long integer is such that you may address its lower order 16 bits as an integer, without error. However, this is highly non-portable (68000 CPU, for example).
- o Non-digits cause the decoding to stop. This includes white space or the end of the string.
- o Use of `atol()` invokes the long integer library.

## b d o s

### The bdos Function

#### Purpose:

Provides execution of basic BDOS functions as described in the Digital Research CP/M 2.2 Interface Guide.

#### Function Header:

```
int bdos(code,DE)
int code;           Function code 0 to 37, 40 (see below)
char *DE;          Parameter, 0 if not used. See below.
```

#### Returns:

Value or error return for the BDOS function. Returns the value in the usual C/80 interface register, which is HL.

#### Example:

```
bdos(11,0);    console status
bdos(1,0);     console input
bdos(2,'A');   print 'A' to the console
bdos(9,"A$");  print string "A" to console
bdos(26,p);    set DMA address to p
bdos(14,1);    log in drive A:
bdos(6,255);   raw console input no-echo
bdos(6,x);    raw character output, char = x
```

WARNING: All arguments must be used. Bdos() requires two arguments of 16-bits each. This interface is not portable. For example, BDS-C does it differently and DRI CP/M-68K uses a LONG 32-bit argument in place of DE or BC. Return values across systems are not consistent. This function is a compromise due to the lack of a standard.

**Notes:**

**BDOS FUNCTION NUMBERS**

0	system reset
1	console input
2	console output, DE=data
3	reader input
4	punch output, DE=data
5	list output, DE=data
6	direct console, DE=255 for input, else output
7	get I/O byte
8	set I/O byte, DE=I/O byte
9	print string terminated with '\$', DE=string address
10	read console buffer (formatted)
11	get console status ( -1 if ready)
12	get version number of CP/M
13	reset disk system
14	select disk drive and log in disk, DE=drive number
15	open existing file, DE=FCB
16	close FCB to disk directory
17	search for first, DE = FCB
18	search for next
19	delete file, DE=FCB
20	read sequential, DE=FCB
21	write sequential, DE=FCB
22	create file, DE=FCB
23	rename file, DE=FCB in special format
24	get login vector, returns HL in special format
25	get current disk, range 0 to 15
26	set DMA address, DE=address for disk operations
27	get address of alloc vector, returns HL
28	write-protect drive
29	get read-only vector, returns HL
30	set file attributes, DE=FCB
31	get address of disk parameter block (DPB), returns HL
32	set or get user code, DE=255 to get, else set code in DE
33	read random, DE=FCB in 36-byte format
34	write random, DE=FCB in 36-byte format
35	compute file size, DE=FCB
36	set random record, DE=FCB in 36-byte format
37	reset drive, DE=drive vector
40	write random with zero fill, DE=FCB

- o Bdos() jumps to the library routine ccBDOS(). The latter allows 1, 2 or 3 arguments. They in turn are not portable, but present a portable interface across machines.

## **binary**

### **The binary Function**

---

#### **Purpose:**

Binary search of a sorted integer array for an integer key value.

#### **Function Header:**

```
int binary(x,v,n)
int x;                      Key value to find in the array v[].
int v[];                     Sorted integer array for lookup.
int n;                      Dimension of the array v[].
```

#### **Returns:**

-1	Failure. The test ( $v[k] == x$ ) failed for $0 \leq k < n$ .
k	Success. A value of TRUE was found for ( $v[k] == x$ ).

**Example:** Look up a terminal type in a table.

```
static char *term[] = {
    "VT52", "ADDS", "TELEVIDEO", "Z29", "H19", "VISUAL200",
    "VISUAL500", "WYSE", "TELERAY", "QUME"
};
static int v[] = {
    5,7,8,9,10,15,16,18,21,25      /* WARNING: MUST BE SORTED! */
};
#include <stdio.h>
main()
{
int k,x;
char s[129];
printf("Possible codes: 5,7,8,9,10,15,16,18,21,25\n");
printf("Enter terminal code: ");
gets(s);
x = atoi(s);
k = binary(x,v,10);
if(k == -1) puts("Not found");
else puts(term[k]);
}
```

## **binary**

**Example:** Print an error message from its error code.

```
static
int v[] = {
    4,12,37,65,101,107,110,201,203,209
};
static
char *err[] = {
    "No disk space","File name invalid","Printer not ready",
    "Invalid character","No disk in drive","Digit expected",
    "Decimal point expected","Exponent expected","Bad drive name",
    "Directory full"
};
#include <stdio.h>
prerror(x)
int x;
{
int k;
    k = binary(x,v,10);
    if(k == -1) puts("Undefined error");
    else puts(err[k]);
}
```

### **Notes:**

- o The assumption that the integer array `v[]` is SORTED is essential.  
It will probably hang the machine if the array is not sorted.
- o A normal application uses sorted data entered in the program by hand as initializers for an integer array `v[]`.
- o To obtain a sorted integer array on the fly, use `numsort()`.

## **Binary Flags**

### **The Binary Flags for C/80**

#### **Purpose:**

Sets I/O mode as ASCII TEXT or BINARY. Files are set by PC\_BIN and GC\_BIN, while the console is set by Cmode.

#### **Function Header:**

extern char Cmode;	Character mode echo = 0 Line mode echo = 1 Character mode no echo = 2
extern char PC_BIN;	ASCII mode = 0 BINARY mode = 1
extern char GC_BIN;	ASCII mode = 0 BINARY mode = 1

#### **Returns:**

Binary mode eliminates all translation. This mode almost simulates UNIX, except for the identity of the newline character.

Ascii text mode throws away carriage return (0x0D) on input and on output translates linefeed to carriage return plus linefeed. In addition, end of file is detected from ctrl-Z. These modes are independent of the open mode, so fill characters at end of file are unchanged from those set at open.

The variable Cmode controls input mode only for the console.

Echo mode prints the character at the time of input. Ascii mode causes echo of carriage return and linefeed for a carriage return only. No echo mode allows input without console echo.

Console Line Mode requires a carriage return to end the input. During input, all CP/M editing features and IObyte functions are in force. Line Mode always echoes.

Console Character Mode is binary input mode for the console. The character is available immediately. Character mode may either echo (Cmode = 0) or not echo (Cmode = 2).

## **B i n a r y F l a g s**

**Example:** Manually set all I/O as binary in order to process foreign data files on a CP/M machine. Then reset the modes for normal processing.

```
#include <stdio.h>
main()
{
extern char PC_BIN,GC_BIN,Cmode;

PC_BIN = GC_BIN = 1; Cmode = 2;
ProcessForeignFiles();
PC_BIN = GC_BIN = 0; Cmode = 1;
printf("Operation completed\n");
}
```

### **Notes:**

- o These features are not present under UNIX nor are they to be expected under most systems. However, simulation of the features is usually possible by writing assembly language interface routines on the host.
- o The function getbyte() uses a mode change Cmode = 2 to get the character. But the value of Cmode is restored after the call.
- o The functions getcbinary() and putcbinary() set and reset the global modes PC\_BIN, GC\_BIN on each call. They may also set and reset Cmode.
- o Console input is buffered to 136 characters under C/80. There is special software in place which loops after each console input in order to pick up type-aheads and function keys. The number of loops is controlled by a variable

```
extern char CCTICK;
```

This variable is currently set at 6, which seems to work well with 3-character function keys and slow auto-repeat. It may not work very well with non-interrupt driven keyboards. Your custom C code may set it to a higher value, if needed.

### The bios Function

---

#### Purpose:

Provides execution of basic BIOS functions as described in the Digital Research CP/M 2.2 Interface Guide.

#### Function Header:

```
int bios(code,BC)
int code;           Function code 1 to 15, see below.
char *BC;           Parameter, 0 if not used. See below.
```

#### Returns:

Value or error returned by the BIOS function, in the normal C/80 interface register, which is HL.

#### Example:

```
x = bdos(2,0);      console status
x = 255 & bios(3,0);  console input
```

WARNING: All arguments must be used. Bios() requires two arguments of 16-bits each. This interface is not portable. For example, BDS-C does it differently and DRI CP/M-68K uses a LONG 32-bit argument in place of DE or BC. Return values across systems are not consistent.

### BIOS FUNCTION NUMBERS

- 1 warm boot
- 2 console status, -1 if ready
- 3 console input, raw
- 4 console output, raw, BC = char
- 5 list output, raw, BC = char
- 6 punch (AX0:) output, raw, BC = char
- 7 reader (AX1:) input, raw
- 8 home disk, BC = drive number
- 9 select disk drive, BC = drive number 0 to 15
- 10 set track, BC = track number 0 to maxtrack
- 11 set sector, BC = sector (usually 1 to maxsector)
- 12 set DMA address, BC = DMA address
- 13 read sector
- 14 write sector, BC = 0 (normal), 1 (directory)
- 15 list status, returns -1 if ready
- 16 sector translate, BC=logical sector, DE=translate table addr  
Must use 3-argument ccBIOS().

- o Bios() jumps to the library routine ccBIOS(). The latter allows 1, 2 or 3 arguments - ccBIOS() is not portable, but presents a portable interface across machines.

## b r k

### The brk Function

---

#### Purpose:

Sets the upper bound of the program to an absolute address.

#### Function Header:

```
int brk(address)
char *address;           Integer return 0 or -1
                        Address to set as new
                        program break address.
```

#### Returns:

0	on success
-1	on failure

**Example:** Reset a C program on exit to its pristine state so that sbrk() sees the same memory on each re-start.

```
extern char *end,*etext,*edata;
brk(end);
```

**Example:** Undo a sequence of calls to sbrk() that have used n bytes of memory, without knowing the base address.

```
unsigned n;
char *sbrk();
brk(sbrk(0)-n);
```

## brk

### Notes:

- o brk() checks for text, data and stack over-write.

- o The symbols end, etext, edata have these meanings:

end	The next usable address after program load.
etext	The physical end of the program text before the file buffers and FCB buffers begin.
edata	Same as etext for C/80 because code and data are in the same place.

- o The externals end, etext, edata are defined in the assembly module ETEXT.C as follows:

```
end: DW $END##+TOT_SZ##
edata: DW $END##
etext: DW $END##
```

- o The double-# marks a symbol as external for M80, e.g., \$END## is equivalent to EXTRN \$END.
- o The object code uses addresses 100h to \$END, that is, the length of the compiled program is (etext-0x100).
- o TOT SZ is the total size of the file buffer and file control block (FCB) area that immediately follows the object code. See STDIO.H. The base address of the area is end (= \$END). The area uses TOT SZ bytes with last address (etext - 1).
- o Reference: For end, edata, etext, see Banahan & Rutter, The Unix Book, Wiley Press (1984), p 198. For a description of brk() and sbrk(), see Banahan, p 205.

## b s e a r c h

### The bsearch Function

#### Purpose:

Binary search of an arbitrary **sorted** table of info. Requires a special compare function to match the internal table structure.

#### Function Header:

```
char *bsearch(key,base,n,w,cmp)
char *key;                      Key for table search.
char *base;                     Base address of sorted table.
int n;                          Table size in elements.
int w;                          Width of each table element.
int (*cmp)();                   Compare function cmp(key,tbl)

int cmp(key,tbl)                User-supplied compare routine.
char *key;                      key = address of the key
char *tbl;                      tbl = address of table entry;
```

#### Returns:

(char *)	Location of table match
(char *)0	Not found

**Example:** Search a sorted list of strings for a key match.

```
#include <unix.h>
#include <stdio.h>
static char *tbl[] = {
    "ABC","AB","EFG","HJK","M","abc","abd","bcf"
};

cmp(p,q) char *p,*q;
{
    return strcmp(p,peekw(q));
}

main()
{
char key[129];
char *p;
char *bsearch();
printf("Enter a 1 to 3 character string: ");
gets(key);
p = bsearch(key,tbl,8,2,cmp);
if(p == (char *)0) puts("Not found");
else puts(peekw(p));
}
```

Note in particular the table width is `sizeof(char *)`. The strings are not all of the same width. The table `tbl[]` is a table of pointers, which causes the unusual addressing modes seen in the example. We used the special function `peekw()` instead of `**-pointers` to document ideas.

## b s e a r c h

### Notes:

- o This is a UNIX function. See also lsearch(), binary(), sbinary(), ssort2(), ssort3(), numsort(), dsort(), dsort16(), shells(), qsort(), quick(), heap().
- o The addressing modes required for string handling are unnatural. See the example above to get it right.
- o The complete source follows, since this is an often used but poorly understood function.

```
char *bsearch(key,base,n,w,cmp)
char *key;
char *base;
int n;
int w;
int (*cmp)();
{
static
char *q,*p;
static
int i,j;
q = base;
while(n > 0) {
    i = n >> 1;
    if((j = (*cmp)(key,p = q + i*w)) == 0) return p;
    else if(j < 0) n = i;
    else {
        q = p + w; n = n - 1 - 1;
    }
}
return ((char *)0);
}
```

## c a l l o c

### The calloc Function

#### Purpose:

Allocates contiguous memory located between the end of the program and the stack, then initializes it to zero.

#### Function Header:

char *calloc(m,s)	The address of the contiguous area
unsigned m,	The number of elements
s;	The size in bytes of each element

#### Returns:

The base address of the area, on success.  
0 if the request fails.

**Example:** Get buffer space for a block read of a data file, then free the space. The file to be read is entered on the command line as:

A>main <datafile

```
#include <unix.h>
#include <stdio.h>
main(argc,argv) int argc; char **argv;
{
int i,n,fd; char *buf;
char *calloc();
buf = calloc(1024,1);
fd = fileno(stdin);
if(buf == (char *)0 || fd == 0) exit();
n = read(fd,buf,1024);
for(i=0;i<n;++i) putchar(buf[i]);
free(buf);
}
```

## c a l l o c

### Notes:

- o The number  $m$  of elements of size  $s$  must be such that  $m*s$  does not overflow as an unsigned integer.
- o The address returned is the base address of a contiguous area of memory whose byte length is  $m*s$ , or 0 for failure.
- o Uses malloc() to get raw system memory.
- o The maximum request is 65531 bytes under CP/M-80.
- o Use free() to release a block. Same as cfree() in the reference manual by Harbison and Steele.
- o Two successive calls to calloc() will not in general result in a single block of contiguous memory.
- o Calls to calloc() cause a header to be written at the beginning of the block. Malloc() uses the header in an essential way - don't write over it!
- o Sfree() and brk() don't know about calloc().

## The ccBDOS Function

---

### Purpose:

Provides execution of basic BDOS functions as described in the Digital Research CP/M 2.2 Interface Guide. The naming conventions attempt to be non-conflicting.

### Function Header:

```
int ccBDOS(DE,BC)
char *DE;           Parameter. May be omitted if not used.
char *BC;           CP/M BDOS function call number.
```

### Returns:

Value or error return for the BDOS function

Bdos calls 12, 24, 27, 29, 31 return an HL vector.

### Examples:

(int)ccBDOS(11);	console status
(int)ccBDOS(1);	console input
(int)ccBDOS('A',2);	print 'A' to the console
(int)ccBDOS("A\$",9);	print string "A" to console
(int)ccBDOS(p,26);	set DMA address to p
(int)ccBDOS(1,14);	log in drive A:
(int)ccBDOS(255,6);	raw console input no-echo
(int)ccBDOS(x,6);	raw character output, char = x

**Notes:****BDOS FUNCTION NUMBERS**

- 0 system reset
- 1 console input
- 2 console output, DE=data
- 3 reader input
- 4 punch output, DE=data
- 5 list output, DE=data
- 6 direct console, DE=255 for input, else output
- 7 get I/O byte
- 8 set I/O byte, DE=I/O byte
- 9 print string terminated with '\$', DE=string address
- 10 read console buffer (formatted)
- 11 get console status (-1 if ready)
- 12 get version number of CP/M
- 13 reset disk system
- 14 select disk drive and log in disk, DE=drive number
- 15 open existing file, DE=FCB
- 16 close FCB to disk directory
- 17 search for first, DE = FCB
- 18 search for next
- 19 delete file, DE=FCB
- 20 read sequential, DE=FCB
- 21 write sequential, DE=FCB
- 22 create file, DE=FCB
- 23 rename file, DE=FCB in special format
- 24 get login vector, returns HL in special format
- 25 get current disk, range 0 to 15
- 26 set DMA address, DE=address for disk operations
- 27 get address of alloc vector, returns HL
- 28 write-protect drive
- 29 get read-only vector, returns HL
- 30 set file attributes, DE=FCB
- 31 get address of disk parameter block (DPB), returns HL
- 32 set or get user code, DE=255 to get, else set code in DE
- 33 read random, DE=FCB in 36-byte format
- 34 write random, DE=FCB in 36-byte format
- 35 compute file size, DE=FCB
- 36 set random record, DE=FCB in 36-byte format
- 37 reset drive, DE=drive vector
- 40 write random with zero fill, DE=FCB

**The ccBIOS Function**

**Purpose:**

Provides execution of basic BIOS functions as described in the Digital Research CP/M 2.2 Interface Guide. The naming conventions attempt to be non-conflicting.

**Function Header:**

```
int ccBIOS(DE,BC,f)
char *DE;           Parameter. May be omitted.
char *BC;           Parameter. May be omitted.
int f;              Function code 1 to 15. Required.
```

**Returns:**

Value or error returned by the CP/M 2.2 BIOS function

Bios calls may return a vector in HL or a byte value.  
The C version always returns in HL.

**Example:**

```
ccBIOS(2);          console status
ccBIOS(3);          console input
ccBIOS('A',4);      print 'A' to the console
ccBIOS(13);         read logical sector of 128 bytes.
```

**Notes:**

**BIOS FUNCTION NUMBERS**

- 1 warm boot
- 2 console status, -1 if ready
- 3 console input, raw
- 4 console output, raw, BC = char
- 5 list output, raw, BC = char
- 6 punch (AX0:) output, raw, BC = char
- 7 reader (AXI:) input, raw
- 8 home disk, BC = drive number
- 9 select disk drive, BC = drive number 0 to 15
- 10 set track, BC = track number 0 to maxtrack
- 11 set sector, BC = sector (usually 1 to maxsector)
- 12 set DMA address, BC = DMA address
- 13 read sector
- 14 write sector, BC = 0 (normal), 1 (directory)
- 15 list status, returns -1 if ready
- 16 sector translate, BC=logical sector, DE=translate table addr  
Must use ccBIOS().

## c c m a i n

### The ccmain Function

#### Purpose:

Standard main program for a compiled C program. Performs setup of the console buffer, command line, stack, heap, re-direction files and passes control to routine main(). Normal return to this function drops through exit().

#### Function Header:

NONE This is not a user-callable function.  
It is used implicitly. See the example below.

#### Returns:

NOTHING ccmain() returns to warm boot.

**Example:** Write a main program in C.

```
#include <stdio.h>
main()
{
    puts("This is the main program, always called main().");
    puts("ccmain is the startup code for main().");
    puts("The end brace of main() causes return to ccmain().");
    puts("Upon return, ccmain() calls exit().");
}
```

**Notes:**

- o The following functions are called by cemain():

<u>Copen</u>	< & > redirection file opener, see COPEN.CC
<u>Cexit</u>	buffer emptying and close files, see CEXIT.CC
<u>tokens</u>	command line processor, see TOKENS.CC
<u>C mcln</u>	move cmd line to Cbuf console buffer, see CMCLN.CC
<u>main</u>	main program, user external

- o The routine uses the following global constants and symbols:

<u>EX SPC</u>	amount of extra space below BDOS, see stdio.h
<u>TOT SZ</u>	size of initial fcb's & file buffers, see stdio.h
<u>\$END</u>	physical end of program, see END.CC

- o The entry points provided in cemain():

<u>VOID exit()</u>	standard exit
<u>VOID a_exit()</u>	abort exit
<u>VOID CTEB()</u>	<sup>^B</sup> jump, language C rules
<u>char *CC CCP</u>	stack pointer for caller of CMAIN
<u>int fin</u>	redirection standard input descriptor
<u>int fout</u>	redirection standard output descriptor
<u>char *p_fin</u>	redirection input filename pointer
<u>char *p_fout</u>	redirection output filename pointer
<u>char *Cbuf</u>	console buffer pointer
<u>char Cmode</u>	char mode echo=0, line mode=1, char mode noecho=2
<u>char *CtlB</u>	<sup>^B</sup> vector address data - extern *char CtlB;

- o The end address of the program is saved in the integer variable \$LM, accessible only through assembly language. This variable is used by sbrk() to manage the heap. The linkage of cemain() to your program is controlled in STDIO.H by declaring this variable as EXTERN.

## C C t l C k

### The CCTICK Variable

#### Purpose:

Contains the number of loops to be made after console input for the purpose of buffering function keys and type-aheads. The default number of additional loops is 6. Adjustment of the variable contents may be needed for slow terminals and non-interrupt CP/M 2.2 console.

#### Function Header:

```
extern char CCTICK;
```

#### Returns:

After a console input, the keyboard is checked for status and ctrl-C and ctrl-B interrupts. This variable controls the number of loops to be made. If a character is available, then it is buffered into the 136-character console buffer and echoed according to the currently set mode flag (0,1 echo but 2 does not).

**Example:** Set up loop cycles of 12 for a slow terminal and CPU which has no interrupt-driven keyboard. This program is designed to capture function keys and display the constituent characters.

```
#include <stdio.h>
main()
{
    extern char CCTICK;
    int x;
    CCTICK = 12;
    puts("Function key test. Press ctrl-Z to exit");
    while( (x = getbyte()) != 26) {
        CtlCk();
        printf("%d Dec, %x Hex, %o Oct, %b Bin\n",x,x,x,x);
    }
}
```

#### Notes:

- o The internal operation of CtlCk() is to call the keyboard status routine to see if the user typed a character. If so, then the character is read no-echo and added to the console buffer. If line mode is active, then ctrl-C, ctrl-B will signal exits to the operating system.
- o In character mode, a call to CtlCk() is the same as adding available characters to the console buffer. No special action is taken for ctrl-C or ctrl-B.

## The ceil Function

### Purpose:

Compute float ceiling, the smallest integer greater than or equal to the given number. For example, ceil(-1.1) = -1, ceil(1.1) = 2.

### Function Header:

```
float ceil(f)
float f;           Float argument.
```

### Returns:

g  
Where g is a whole number, signed,  
with  $f \leq g$  and  $g-1 < f$ .

### Example:

```
#define pfFLOAT 1
#include <math.h>
#include <stdio.h>
main(argc,argv) int argc; char **argv;
{
char s[129];
float f;
float atof();
float ceil();
printf("Enter float f: "); gets(s); f = atof(s);
printf("ceil(%f) = %f\n",f,ceil(f));
}
```

### Notes:

- o No error codes returned. No need to code for errno.
- o Ceil() does not truncate. For  $x > 0$ , add 1 and truncate, but for  $x < 0$  truncate.
- o No overflow check.
- o No auto conversion to FLOAT.

## **C e x i t \_**

### **The Cexit \_ Function**

---

#### **Purpose:**

Flushes buffers and closes the file control block to the disk directory for stream stdout.

#### **Function Header:**

```
VOID Cexit_()
```

#### **Returns:**

Nothing.

**Example:** Making a new Cexit () to simulate systems that flush and close all files on exit.

```
Cexit_()  
{  
    extern char IOch[];  
    extern int MAXCHN[];  
    int i,fd;  
  
    for(i=0;i<MAXCHN;++i) {  
        if((fd = IOch[i]) != -1) fclose(fd);  
    }  
}
```

#### **Notes:**

- o The provided Cexit () routine flushes and closes stream stdout. But any other open streams will be ignored. Open files will leave orphan disk directory entries with 0 records.

## The cfs Function

---

### Purpose:

Compute the virtual file size of a CP/M disk file.

### Function Header:

```
unsigned cfs(file)
char *file;           Normal CP/M file name. May not be a
                      device name.
```

### Returns:

The file size in 128-byte records. This may not be the right answer if the file was fragmented during creation or if the file is currently open.

**Example:** Find out the length in bytes of a binary file.

```
#define pfLONG 1
#include <stdio.h>
main()
{
    long n;
    unsigned x,cfs();
    char s[129];
    printf("Enter a file name: ");
    gets(s);
    x = cfs(s);
    n = x*128L;
    if(x != -1)
        printf("%s has length %ld bytes\n",s,n);
    else
        printf("File %s not found\n",s);
}
```

### Notes:

- o Here is the source code for cfs().

```
unsigned int cfs(file)
char *file;
{
    auto char fcb[36];
    char *findFIRST();
    makeFCB(fcb,file);
    if(findFIRST('V',file) == (char *)0) return -1;
    ccBOOS(fcb,35);
    return peekw(fcb+33);
}
```