RED
A Better C Screen Editor, Part II

Last month I introduced RED, a new screen editor which is written in Small-C. That article covered RED's buffering scheme in great detail. This month I'll discuss the Small-C Library used in RED and improvements that might be made to RED. I'll also say a few words about why RED is copyrighted.

To the user, RED appears very similar to the ED2 editor described in the January 1982 issue of *Dr. Dobb's Journal*. Because of this, the remainder of RED is provided in Listing One (page 66) without extensive discussion. Tables 1, 2 (at right), and 3 (page 64) give a brief summary of RED for those not familiar with the original editor.

A New Run-Time Library for Small-C

The buffer routines described last month require a run-time library which supports "unbuffered," i.e., block-at-a-time, file access. The new library presented in Listing Two (page 87) is an edited and slightly modified version of part of the BDS C run-time library. (There is a lot more to the library that I have not included.) This library is presented here by permission of its author, Leor Zolman.

I have tried to make sure that the routines presented here work *exactly* the same as the original BDS C library routines. Changes to the code were made only where absolutely necessary and are marked in the listing. The changes were made because Small-C pushes arguments on the stack in reverse of the order that BDS C uses. Small-C pushes parameters in the order in which they appear in the parameter list. Thus, the last argument in the list appears at the *top* of the stack (right under the return address). In BDS C, the *first* argument appears at the top of the stack.

The new library passes command line arguments to the main() function. To retrieve these arguments, add two arguments to main() as follows:

by Edward K. Ream

Edward K. Ream, 1850 Summit Avenue, Madison, Wisconsin 53705.

Table 1
COMMAND MODE

append *file* . . . . . Append *file* after the current line.
change *line range* . . . . . Change lines which match a pattern.
? in search mask matches any character.
? in change mask becomes character that matched corresponding ? in the search mask.
^ in search mask in column 1 matches the start of a line.
clear . . . . . Clear the buffer.
copy *line range number* . Copy lines in *line range* after line *number*.
delete *line range* . . . . . Delete ALL lines in *line range*.
dos . . . . . Exit from RED.
find . . . . . Go to next line that matches a pattern.
? and ^ as in change command.
g *number* . . . . . Go to line *number*.
help . . . . . Print a help message for command mode.
list *line range* . . . . . List lines to the printer.
load *file* . . . . . Load the buffer from a file.
move *line range number* . Move lines in *line range* after line *number*.
name *filename* . . . . . Make *filename* the current file name.
resave . . . . . Save the buffer to an existing file.
save . . . . . Save the buffer to a new file.
search *line range* . . . . . Search for a pattern.
? and ^ as in change command.
tabs *number* . . . . . Set tab stops to every *number* column.

Table 2
SPECIAL CHARACTERS

Key	Default Value	Result
up key:	control-u . .	Move the cursor up. Enter edit mode.
down key:	control-d . .	Move the cursor down. Enter edit mode.
right key:	control-r . . .	Move the cursor right.
left key:	back space . .	Move the cursor left.
insert up key:	LF	Insert a new line above the current line. Enter insert mode.
insert down key:	CR	Insert a new line below the current line. Enter insert mode.
delete character key:	DEL	Delete one character.
delete line key:	control-z . .	Delete the current line.
insert key:	control-n . .	Enter insert mode.
command key:	escape	Enter command mode.
edit key:	control-e . .	Enter edit mode.
undo key:	control-x . .	Undo changes to the current line.
split key:	control-s . . .	Split the current line.
join key:	control-p . .	Join current line, preceding line.
repeat key:	conrol-a . . .	Repeat the previous character.

```
main(argc, argv)
int argc;
int * argv;
  /* Kludge: should be char ** argv */
```

Argc is *one more than* the number of arguments on the command line. Argv is a pointer to an array of pointers to the actual arguments. Argv [0] is not used, thus argv [1] is a pointer to the first argument. Because of the limitations in Small-C, argv must be declared as shown.

The library also provides the following unbuffered I/O primitives. These *primitives work* just the same as the primitives in the BDS C Library.

close (fd)
int fd;

Closes a file which was opened by open() or creat(). Returns -1 on an error.

creat (filename)
char * filename;

Creates an empty file. The file is erased if it exists. The file is opened for reading or writing. Returns a file descriptor (a short integer) which is used by the read(), write(), seek(), tell(), close(), and fabort() routines.

fabort (fd)
int fd;

Frees the file descriptor fd without closing the file. Don't use the fabort() on a file which was open for writing unless you are willing to lose data which were written into it.

open (filename, mode)
char * filename;
int mode;

Opens a file for reading (mode 0), writing (mode 1), or both reading and writing (mode 2). The file must already exist; -1 is returned if it doesn't. Returns a file descriptor on a successful open.

read (fd, buffer, nbl)
int fd;
char * buffer;
int nbl;

Reads nbl sectors of the file whose file descriptor is fd into the buffer. All sectors are 128 bytes in the CP/M world, regardless of their actual size on your disk. Returns the number of blocks actually read, or -1 for errors. 0 means end of file.

rename (oldname, newname)
char * oldname, * newname;

Renames the file.

seek (fd, n, code)
int fd, n, code;

Positions the file for reading or writing at a particular sector. Positions the file at the nth sector if code is 0. Positions the file n sectors past the present sector if code is 1. Returns -1 on a seek for a nonexistent sector. Make sure to close the file if seek fails; the file is sure to be in bad shape.

tell (fd)
int fd;

Returns the current position of the file (an integer). The first sector of the file is sector 0.

write (fd, buffer, nbl)
int fd;
char * buffer;
int nbl;

Writes nbl sectors from the buffer to the file. Returns -1 on an error, which is usually caused by the disk becoming full.

unlink (filename)
char * filename;

Erases the file (permanently!) from the disk. Use with caution.

Finally, the new library provides two functions which allow jumps between procedures. This is a very important capability to have for error recovery.

setjmp (buffer)
char buffer [6];

longjmp (buffer, val)
char buffer [6];
int val;

Setjmp() saves the current state of the processor in the buffer. (For Small-C this means the BC register, the program counter, and the stack pointer.) Setjmp() itself always returns 0, but see below. Whenever a later call to longjmp() is made (from *anywhere* in the current function or a lower-level function) the CPU state is restored to what it was when setjmp() was called last with the same buffer argument. The program behaves as if control were returning from the setjmp() function *except* that the return value is the "val" which was passed to longjmp().

Ideas for Improving the Editor

Nothing is impossible for the man who doesn't have to do it himself.

— Weiler's Law

The algorithms presented last month are the best that I know of; I would like to hear about any improvements that you might have. Of course, improvements to the editor are possible. Here is a short list.

Rewriting functions in assembly language is an obvious way to increase speed by 10% to 100%. The swap_in(), b_scan(), and bufgetln() routines would be my candidates for recoding because they are used so often. It is likely that other routines would also benefit from being written in assembly language.

Another idea is to change the format of the data area of a block. Instead of preceding each line with a count, it is possible to create a table of pointers to the start of each line. This table would make searching in the b_scan() routine unnecessary. Dave Cortesi discussed this idea and others at length in Dr. Dobb's Clinic (*DDJ* No. 65, March 1982, pages 6-9). Putting such a table in each block will slow down the insert and delete operations because the table must be adjusted. However, that penalty is slight and the gain to be had from speeding up b_scan() is much larger.

Perhaps the most interesting improvement to the editor would be to add multiple windows to the screen. The new buffer routines could be modified easily to deal with multiple files. The idea is to have all files share the same set of buffers; all that must be done to change from one file to another is to swap out all the dirty blocks. Of course, such a change would require a major revision to the window module on file red4.sc.

RED and Copyright

I think it is important to distribute source code for virtually all programs. Just as I used the code in the "Just Like Mom's" editor as a starting point, I hope some of you will take these routines and use them in new ways.

RED is copyrighted solely to protect myself against those who distribute RED without permission. I hope you will feel free to do anything with this editor except distribute it for profit. Please don't let the copyright prevent you from using the code; if you are unsure about how I may react to your plans, just give me a call. I'm sure we can work something out.

Acknowledgements

RED owes much to my friends at the Waisman Center of the University of Wisconsin, Madison; Cliff Gilman greatly improved RED's documentation, Bruce Orchard assisted in moving RED to the Harris computer, and Dave Wilson suggested subtle, but important, improvements in the workings of various commands. ■■

(Listing One begins on page 66)

(Listing Two begins on page 87)

RED - Listing (Text begins on page 62)

Listing One

```
/*
        RED main program -- small-C version

        Source:  red2.sc
        Version: January 22, 1983

        Copyright (C) 1983 by Edward K. Ream
*/

/* Define the disk recovery point. */

char D_ERROR [6];

/* the main program dispatches the routines that
 * handle the various modes.
 */

#define CMNDMODE 1      /* enter command mode flag */
#define INSMODE  2      /* enter insert modes flag */
#define EDITMODE 3      /* enter edit mode flag */
#define EXITMODE 4      /* exit editor flag */

main()
{

        int mode;

        /* ready the system module */
        sysinit();
        /* clear the main buffer */
        bufnew();
        /* fmt output by default goes to screen */
        fmtasss(NO);
        /* set tabs, clear the screen and sign on */
        fmtset(8);
        outclr();
        outxy(0,SCRNL1);
        message(SIGNON);
        message(VERSION);
        message(COPYRIGHT);
        message("");
        message(XSIGN);
        message(XSIGN1);
        outxy(0,1);
        /* clear filename [] for save(), resave() */
        pmtclr();
        /* start off in command mode */
        mode=CMNDMODE;


        /* get null line 1 for edit() */
        edgetln();
        while(1){
                if (mode == EXITMODE) {
                        break;
                }
                else if (mode == CMNDMODE) {
                        mode=command();
                }
                else if (mode == EDITMODE) {
                        mode=edit();
                }
                else if (mode == INSMODE) {
                        mode=insert();
                }
                else {
                        syserr("main: no mode");

                        mode=EDITMODE;
                }
        }
}

/*
 * handle edit mode.
 * dispatch the proper routine based on one-character commands.
 */

edit()
{
char buffer [SCRNW+1];
int v;
int x,y, topline;
char c;
        /* we can't do edgetln() or edgo() here because
         * those calls reset the cursor.
         */

        /* Set disk error recovery point. */
        setjmp(D_ERROR);

        pmtedit();
        while(1){
                /* get command */
                c=tolower(syscin());

                /* comment out ----- 9/14/82
                if (c == 3) {
                        bufdump();
                }
                ----- end comment out */

                if ( (c == ESC1) | (c=='c') ) {
                        /* enter command mode. */
                        return CMNDMODE;
                }
                else if ( (c == INS1) | (c=='i') ) {
                        /* enter insert mode */
                        return INSMODE;
                }
                else if (special(c) == YES) {
                        if ( (c == UP1) | (c == DOWN1) ) {
                                return INSMODE;
                        }
                        else {
                                continue;
                        }
                }
                else if (control(c) == YES) {
                        continue;
                }
                else if (c == ' ') {
                        edright();
                        pmtcol();
                }
                else if (c == 'b') {
                        edbegin();
                        pmtcol();
                }
                else if (c == 'd') {
                        /* scroll down */
                        pmtmode("edit: scroll");
                        sysswait();
                        while (bufnrbot() == NO) {

                                if (chkkey() == YES) {
                                        break;
                                }
                                eddn();
                        }
                        pmtedit();
                }
                else if (c == 'e') {
                        edend();
                        pmtcol();
                }
                else if (c == 'g') {
                        /* save x,y in case don't get number */
                        x=outgetx();
                        y=outgety();
                        pmtcmnd("edit: goto: ",buffer);
                        if(number(buffer,&v)) {
                                edgo(v,0);
                        }
                        else {
                                outxy(x,y);
                        }
                        pmtedit();
                }

                else if (c == 'h') {
                        /* remember how screen was drawn */
                        x=outgetx();
                        y=outgety();
                        topline=bufln()-y+1;

                        /* output the help message */
                        outclr();
                        outxy(0,SCRNL1);
                        edithelp();

                        /* redraw the screen */
                        bufout(topline,1,SCRNL1);
                        outxy(x,y);
                        pmtedit();
                }

                else if (c == 'k') {
                        pmtmode("edit: kill");
                        c=syscin();
                        if ( (special(c) == NO) &
                             (control(c) == NO)
                           ) {
                                edkill(c);
                        }
                        pmtedit();
                }
                else if (c == 's') {
                        pmtmode("edit: search");
                        c=syscin();
                        if ( (special(c) == NO) &
                             (control(c) == NO)
                           ) {
                                edsrch(c);
                        }
                        pmtedit();
                }
                else if (c == 'u') {
                        /* scroll up */
                        pmtmode("edit: scroll");
                        sysswait();
                        while (bufattop() == NO) {
                                if (chkkey() == YES) {
                                        break;
                                }
                                edup();
                        }
                        pmtedit();
                }
                else if (c == 'x') {
                        pmtmode("edit: eXchange");
                        c=syscin();
                        if ( (special(c) == NO) &
                             (control(c) == NO)
                           ) {
                                edchng(c);
```

```
                        }
                        pmtedit();
                }
                /* do nothing if command not found */
        }
}

/* insert mode.
 * in this mode the UP1, UP2 keys reverse their roles,
 * as do the DOWN1, and DOWN2 keys.
 */

insert()
{
char c;
        /* Set disk error recovery point. */
        setjmp(D_ERROR);

        pmtmode("insert");

        while (1) {
                /* get command */
                c=syscin();
                if (c == ESC1) {
                        /* enter command mode */
                        return CMNDMODE;
                }
                else if (c == EDIT1) {
                        /* enter edit mode */
                        return EDITMODE;
                }
                else if (c == INS1) {
                        /* do nothing */
                        ;
                }
                else if (special(c) == YES) {
                        if ( (c == UP2) || (c == DOWN2) ) {
                                return EDITMODE;
                        }
                        else {
                                continue;
                        }
                }
                else if (control(c) == YES) {
                        /* ignore non-special control chars */
                        continue;
                }
                else {
                        /* insert one char in line */
                        edins(c);
                        pmtcol();




                }
        }
}

/* return YES if c is a control char */

control(c) char c;
{
        if (c == TAB) {
                return NO;      /* tab is regular */
        }
        else if (c>=127) {
                return YES;     /* del or high bit on */
        }
        else if (c < 32) {
                return YES;     /* control char */
        }
        else {
                return NO;      /* normal */
        }
}

/*
 * handle the default actions of all special keys.
 * return YES if c is one of the keys.
 */

special(c) char c;
{
int k;
        if (c == JOIN1) {
                edjoin();
                pmtline();
                return YES;
        }
        if (c == SPLT1) {
                edsplit();
                pmtline();
                return YES;
        }
        if (c == ABT1) {
                edabt();
                pmtcol();
                return YES;
        }
        else if (c == DEL1) {
                eddel();
                pmtcol();
                return YES;
        }
        else if (c == ZAP1) {
                edzap();
                pmtline();
                return YES;
        }
        else if (c == UP2) {
                /* move up */
                edup();
                pmtline();
                return YES;
        }
        else if (c == UP1) {
                /* insert up */
                ednewup();
                pmtline();
                return YES;
        }
        else if (c == DOWN2) {
                /* move down */
                eddn();
                pmtline();
                return YES;
        }
        else if (c == DOWN1) {
                /* insert down */
                ednewdn();
                pmtline();
                return YES;
        }
        else if (c == LEFT1) {
                edleft();
                pmtcol();
                return YES;
        }
        else if (c == RIGHT1) {
                edright();
                pmtcol();
                return YES;
        }
        else {
                return NO;
        }
}

/*
 * command() dispatches command routines while
 * in command mode.
 */

command()
{
int v;
char c;
char args[SCRNW1];
char *argp;
int topline;
int ypos;
int oldline;
int k;
        /* command mode commands may move the current line.
         * command mode must save the current line on entry
         * and restore it on exit.
         */
        edrepl();
        /* remember how the screen was drawn on entry */
        oldline=bufln();
        ypos=outgety();
        topline=oldline-ypos+1;

        /* Set disk error recovery point. */
        setjmp(D_ERROR);

        while(1) {
                syswait();
                outxy(0,SCRNL1);
                fmterlf();
                pmtmode("command:");
                getcmnd(args,0);
                fmterlf();
                pmtline();
                c=args[0];
                if ( (c == EDIT1) || (c==INS1) ) {
                        /* redraw screen */
                        if (oldline == bufln()) {
                                /* get current line */
                                edgetln();
                                /* redraw old screen */
                                bufout(topline,1,SCRNL1);
                                outxy(0,ypos);
                                syswait();
                        }
                        else {
                                /* update line and screen */
                                edgo(bufln(),0);
                                syswait();
                        }
                        if (c == EDIT1) {
                                return (EDITMODE);
                        }
                        else {
                                return (INSMODE);
                        }
                }
                else if (tolower(args[0]) == 'g'){
                        argp=skipbl(args+1);
                        if (argp[0] == EOS) {
                                edgo(oldline,0);
                                return EDITMODE;
                        }
                        else if (number(argp,&v) == YES) {
                                edgo(v,0);
                                return EDITMODE;
                        }
                        else {
                                message("bad line number");
                        }
                }
```

(Continued on page 70)

RED - Listing

(Listing continued, text begins on page 62)

```
                else if (lookup(args,"append")) {
                        append(args);
                }
                else if (lookup(args,"change")) {
                        change(args);
                }
                else if (lookup(args,"clear")) {
                        clear();
                }
                else if (lookup(args, "copy")) {
                        copy(args);
                }
                else if (lookup(args,"delete")) {
                        delete(args);
                }
                else if (lookup(args,"dos")) {
                        if (chkbuf() == YES) {
                                /* clean up any temp files. */
                                bufend();
                                return (EXITMODE);
                        }
                }
                else if (lookup(args,"find")) {
                        if ((k = find()) >= 0) {
                                edgo(buflin(),k);
                                return EDITMODE;
                        }
                        else {
                                /* get current line */
                                bufgo(oldline);
                                edgetln();

                                /* stay in command mode */
                                message("pattern not found");
                        }
                }
                else if (lookup(args, "help")) {
                        help();
                }
                else if (lookup(args,"list")) {
                        list(args);
                }
                else if (lookup(args,"load")) {
                        load(args);
                }
                else if (lookup(args,"move")) {
                        move(args);
                }
                else if (lookup(args,"name")) {
                        name(args);
                }
                else if (lookup(args,"resave")) {
                        resave();
                }
                else if (lookup(args,"save")) {
                        save();
                }
                else if (lookup(args,"search")) {
                        search(args);
                }
                else if (lookup(args,"tabs")) {
                        tabs(args);
                }
                else if (lookup(args,"")) {
                        ;
                }
                else {
                        message("command not found");
                }
        }
}

/* return YES if line starts with command */

lookup(line,command) char *line, *command;
{
        while(*command) {
                if (tolower(*line++) != *command++) {
                        return NO;
                }
        }
        if((*line == EOS) | (*line == ' ') | (*line == TAB)) {
                return YES;
        }
        else {
                return NO;
        }
}

/* get next command into argument buffer */

getcmd(args,offset) char *args; int offset;
{
int j,k;
char c;
        outxy(offset,outgety());
        outdeol();
        k=0;

        while ((c=sysgin()) != CR) {
                if ( (c == EDIT1) | (c == INS1) ) {
                        args[0]=c;
                        return;
                }
                if ( (c == DEL1) | (c == LEFT1) ) {
                        if (k>0) {
                                outxy(offset,outgety());
                                outdeol();
                                k--;
                                j=0;
                                while (j < k) {
                                        outchar(args[j++]);
                                }
                        }
                }
                else if (c == ABT1) {
                        outxy(offset,outgety());
                        outdeol();
                        k=0;
                }
                else if ((c != TAB) & ((c < 32)|(c == 127))) {
                        /* do nothing */
                        continue;
                }
                else {
                        if (k+offset < SCRNW1) {
                                args[k++]=c;
                                outchar(c);
                        }
                }
        }
        args[k]=EOS;
}
```

```
/*
        RED command mode commands -- small-C version

        Source:  red3.sc
        Version: January 23, 1983;  February 26, 1983

        Copyright (C) 1983 by Edward K. Ream
*/

/* Define data global to these routines. */

char filename[SYSFNMAX];


/*
        Append command.
        Load a file into main buffer at current location.
        This command does NOT change the current file name.
*/

append(args)
char *args;
{
        char buffer[MAXLEN];   /* disk line buffer */
        int file;
        int n;
        int topline;
        char locfn[SYSFNMAX];  /* local file name */

        /* Get file name which follows command. */
        if (namel(args,locfn) == ERR) {
                return;
        }
        if (locfn[0] == EOS) {
                message("no file argument");
                return;
        }

        /* Open the new file. */
        if ((file = sysopen(locfn, 0)) == ERR) {
                message("file not found");
                return;
        }

        /* Read the file into the buffer. */
        while ((n=sysrdln(file,buffer,MAXLEN)) >= 0) {
                if (n > MAXLEN) {
                        message("line truncated");
                        bufdump();
                        exit();
                        n=MAXLEN;
                }
                bufins(buffer,n);
                bufdn();
        }

        /* Close the file. */
        sysclose(file);

        /*
                Redraw the screen so topline will be at top
                of the screen after command() does a CR/LF.
        */
        topline=max(1,buflin()-SCRNL2);
        bufout(topline,2,SCRNL2);
        bufgo(topline);
}


/* Global change command. */

change(args)
char *args;
{
        char oldline[MAXLEN1]; /* reserve space for EOS */
        char newline[MAXLEN1];
        char oldpat[MAXLEN1];
        char newpat[MAXLEN1];
        int from, to, col, n, k;

        /* Check the arguments. */
        if (get2args(args,&from,&to) == ERR) {
                return;
        }

        /* get search and change masks into oldpat, newpat */
        fmtsout("search mask ?  ",0);
        getcmd(oldpat,15);
        fmtcrlf();
```

```
        if (oldpat[0] == EOS) {
                return;
        }
        pmtline(":");
        fmtsout("change mask ?  ",0);
        getcmd(newpat,15);
        fmtcrlf();

        /* make substitution for lines between from, to */
        while (from <= to) {
                if (chkkey() == YES) {
                        break;
                }
                bufgo(from++);
                if (bufatbot() == YES) {
                        break;
                }
                n=bufgetln(oldline,MAXLEN);
                newline[n]=EOS;
                oldline[n]=EOS;

                /* '^' anchors search */
                if (oldpat[0] == '^') {
                        if (amatch(oldline,oldpat+1,0) == YES) {
                                k=replace(oldline,newline,
                                        oldpat+1,newpat,0);
                                if (k == ERR) {
                                        return;
                                }
                                fmtcrlf();
                                putdec(bufln(),5);
                                fmtsout(newline,5);
                                outdeol();
                                bufrepl(newline,k);
                        }
                        continue;
                }

                /* search oldline for oldpat */
                col=0;
                while (col < n) {
                        if (amatch(oldline,oldpat,col++) == YES){
                                k=replace(oldline,newline,
                                        oldpat,newpat,col-1);
                                if (k == ERR) {
                                        return;
                                }
                                fmtcrlf();
                                putdec(bufln(),5);
                                fmtsout(newline,5);
                                outdeol();
                                bufrepl(newline,k);
                                break;


                        }
                }
        }
        fmtcrlf();
}

/* clear main buffer and file name */

clear()
{
        /* make sure it is ok to clear buffer */
        if (chkbuf() == YES) {
                filename[0]=0;
                pmtfile("");
                outclr();
                outxy(0,SCRNLI);
                bufnew();
                message("buffer cleared");


        }
}

/* Block copy command. */

copy(args)
char * args;
{
        int i, k;
        int last;
        int fstart, fend, tstart;
        char buffer[MAXLEN+1];

        /* Get exactly three args. */
        if (get3args(args, &fstart, &fend, &tstart) == ERR) {
                return;


        }

        /*
                The 'to' and 'from' blocks must not overlap.
                Fstart must be > 0, tstart must be >= 0.
        */
        if ( (fend < fstart) |
             (fstart <= 0)   |
             (tstart < 0)    |
             ( (tstart >= fstart) & (tstart < fend) )
           ) {
                message("oops,  check the copy parameters");
                return;
        }

        /* Make sure the last line exists. */
        last = max(tstart, fstart);

        bufgo(last);

        if (bufln() != last) {
                message("Last line doesn't exist.");
                return;
        }

        /*
                Move the 'from block' to the 'to block'.
                Move one line at a time.
        */
        i = 0;
        while (i <= fend - fstart) {

                /* Go to next line of 'from block'. */
                if (fstart < tstart) {
                        bufgo(fstart + i);
                }
                else {
                        bufgo(fstart + fend - i);
                }

                if (bufatbot()) {
                        /* end of 'from block' */
                        break;
                }

                /* Get line of 'from block' into buffer. */
                k = bufgetln(buffer, MAXLEN);

                /* Go to next line of 'to block'. */
                bufgo(tstart + i + 1);

                /* Insert next line into 'to block'. */
                bufins(buffer, k);

                /* Bump the count. */
                i++;
        }
}


/* multiple line delete command */

delete(args)
char *args;
{
        int from, to;

        /* Check the request. */
        if (get2args(args,&from,&to) == ERR) {
                return;
        }
        if (from > to) {
                return;
        }

        /* go to first line to be deleted */
        bufgo(from);

        /* delete all lines between from and to */
        bufdeln(to-from+1);

        /* redraw the screen */
        bufout(bufln(),1,SCRNLI);
}

edithelp()
{

message(
"Here is a list of the commands that you can use in edit mode."
); message(
"Control characters (preceeded by ^) may also be used in insert mode."
); message(
"Type help when in command mode for a list of command mode commands."
); message(
""
); message(
"b          beginning of line    ^d       move cursor up; enter edit mode"
); message(
"c or ESC   enter command mode   ^e       enter edit mode"
); message(
"d          scroll down          ^h       delete character"
); message(
"e          end of line          ^l       move cursor left"
); message(
"g <n>      go to line <n>       ^p       join two lines"
); message(
"h          print this message   ^r       move cursor right"
); message(
"i or ^n    enter insert mode    ^s       split line at cursor"
); message(
"k <let>    delete to <let>      ^u       move cursor up; enter edit mode"
); message(
"s <let>    set cursor to <let>  ^x       undo changes to the line"
); message(
"u          scroll up            ^z       delete line"
); message(
"carriage return:   insert line below current line;  enter insert mode"
); message(
"line feed:         insert line above current line;  enter insert mode"
); message(
""
); message(
"Type any character to continue editing... "
);
        pmtedit();
        syscin();
}
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
 * search all lines below the current line for a pattern
 * return -1 if pattern not found.
 *        ...   ...    column number of start of pattern.
 *


                        ln() + 1, HUGE, YES);


help()

message(
"Here is a list of commands you can use in command mode."
); message(
"Type ? when in edit mode for more help."
); message(
""
); message(
"append <filename>     append a file after the current line"
); message(
"change <line range>   change all lines in <line range>"
); message(
"clear                 reset the editor"
); message(
"copy <n1> <n2> <n3>   copy lines <n1> through <n2> after <n3>"
); message(
"delete <line range>   delete all lines in <line range>"
); message(
"exit                  exit from the editor"
); message(
"find                  search for a pattern;  Enter edit mode if found"
); message(
"g <n>                 enter edit mode at line <n>"
); message(
"g or 'g               enter edit mode at the current line"
); message(
"help                  print this message"
); message(
"list                  list lines to the printer"
); message(
"load <filename>       replace the buffer with <filename>"
); message(
"move <n1> <n2> <n3>   move lines <n1> through <n2> after <n3>"
); message(
"name <filename>       set the filename for the save and resave"
); message(
"resave                save the buffer to the already existing file"
); message(
"save                  save the buffer to a new file"
); message(
"search                list all lines which contain a pattern"
); message(
"tabs <n>              set tabs to every <n> columns"
);


/* list lines to list device */

list(args)
char *args;
{
        char linebuf[MAXLEN1];
        int c;
        int from, to, line, oldline;

        /* save the buffer's current line */
        oldline=bufln();

        /* get starting, ending lines to print */
        if (get2args(args,&from,&to) == ERR) {
                return;
        }

        /* print lines one at a time to list device */
        line=from;
        while (line <= to) {

                /* make sure prompt goes to console */
                fmtasn(NO);

                /* check for interrupt */
                if (chkkey() == YES) {

                        break;
                }

                /* print line to list device */
                fmtasn(YES);

                bufgo(line++);
                if (bufatbot()) {
                        break;
                }
                nebufgetln(linebuf,MAXLEN1);
                nsmln(n,MAXLEN);
                linebuf[n]=CR;
                fmtsout(linebuf,0);
                fmtcrlf();
        }

        /* redirect output to console */
        fmtasn(NO);

        /* restore cursor */
        bufgo(oldline);
}


/* load file into buffer. */

load(args)
char *args;
{
        char buffer[MAXLEN];    /* disk line buffer */
        char locfn[SYSFNMAX];   /* file name */
        int n;
        int topline;

        /* Get filename following command. */
        if (name1(args,locfn) == ERR) {
                return;
        }

        if (locfn[0] == EOS) {
                message("no file argument");
                return;
        }

        /* Give user a chance to save the buffer. */
        if (chkbuf() == NO) {
                return;
        }

        /* Open the new file. */
        if (sysexists(locfn) == NO) {
                message("file not found");
                return;
        }

        /* Update file name. */
        syscopfn(locfn, filename);
        pmtfile(filename);

        /* Clear the buffer. */
        bufnew();

        /* Read the whole file into the buffer. */
        buf_f_file(filename);
        /* indicate that the buffer is fresh */
        bufsaved();

        /* set current line to line 1 */
        bufgo(1);

        /*
                Redraw the screen so that topline will be
                on line 1 after command() does a CR/LF.
        */
        topline=max(1,bufln()-SCRNL2);
        bufout(topline,2,SCRNL2);
        bufgo(topline);
}


/* Block move command. */

move(args)
char * args;
{
        int c, i, k;
        int last;
        int fstart, fend, tstart;
        char buffer[MAXLEN1];

        /* Get exactly three args. */
        if (get3args(args, &fstart, &fend, &tstart) == ERR) {
                return;
        }

        /*
                The 'to' and 'from' blocks must not overlap.
                Fstart must be > 0, tstart must be >= 0.
        */
        if ( (fend < fstart)  ||
             (fstart <= 0)    ||
             (tstart < 0)     ||
             ( (tstart >= fstart) & (tstart <= fend) )
           ) {
                message("oops, check the move parameters");
                return;
        }

        /* Make sure the last line exists. */
        if (tstart < fstart) {
                last = fstart;
        }
        else {
                last = tstart;
        }

        bufgo(last);
        if (bufln() != last) {
                return;
        }

        /*
                Move the 'from block' to the 'to block'.
                Move one line at a time.
        */
        i = c = 0;
        while (c++ <= fend - fstart) {

                /* Go to next line of 'from block'. */
                bufgo(fstart + i);

                if (bufatbot()) {
                        /* end of 'from block' */
                        break;
                }
```

```
                /* Get line of 'from block' into buffer. */
                k = bufgetln(buffer, MAXLEN);

                /* Delete this line. */
                bufdeln(1);
                /* Go to next line of 'to block'. */
                if (tstart < fstart) {

                        /* Delete leaves 'to block' numbers. */
                        bufgo(tstart + i + 1);
                }
                else {

                        /* Delete decreased numbers by one. */
                        bufgo(tstart + i);
                }

                /* Insert next line into 'to block'. */
                bufins(buffer, k);

                /* Adjust line numbers if needed. */
                if (tstart < fstart) {

                        /* Line numbers increase. */
                        i++;
                }
        }
}


/* change current file name */

name(args)
char *args;
{
        name1(args,filename);
        pmtfile(filename);
}


/* check syntax of args.
 * copy to filename.
 * return OK if the name is valid.
 */

name1(args,filename)
char *args, *filename;
{
        /* skip command */
        args=skiparg(args);
        args=skipbl(args);

        /* check file name syntax */
        if (syschkfn(args) == ERR) {
                return ERR;
        }

        /* copy filename */
        syscopfn(args,filename);

        return OK;
}


/* Save the buffer in an already existing file. */

resave()
{
        int n, oldline;

        /* Save line number. */
        oldline = bufln();

        /* Make sure file has a name. */
        if (filename[0] == EOS) {
                message("file not named");
                return;
        }

        /* The file must exist for resave. */
        if (sysexists(filename) == NO) {
                message("file not found");
                return;
        }

        /* Write out the whole buffer. */
        buf_w_file(filename);

        /* Indicate that the buffer has been saved. */
        bufsaved();

        /* Restore line number. */
        bufgo(oldline);
}
/* Save the buffer in a new file. */

save()
{
        int file, n, oldline;

        /* Save current line number. */
        oldline = bufln();

        /* Make sure the file is named. */
        if (filename[0] == EOS) {
                message("file not named");
                return;
        }

        /* File must NOT exist for save. */
        if (sysexists(filename) == YES) {
                message("file exists");
                return;
        }

        /* Write out the whole buffer. */
        buf_w_file(filename);

        /* Indicate buffer saved. */
        bufsaved();

        /* Restore line number. */
        bufgo(oldline);
}
/* global search command */

search(args)
char *args;
{
int from, to;

        /* Check the request. */
        if (get2args(args,&from,&to) == ERR) {
                return;
        }
        search1(from, to, NO);
}

/* search lines for a pattern.
 * if flag  ==  YES: stop at the first match.
 *                   return -1 if no match.
 *                   otherwise return column number of match.
 * if flag  ==  NO:  print all matches found.
 */

search1(from, to, flag)
int from, to, flag;
{
        char pat   [MAXLEN1];   /* reserve space for EOS */
        char line  [MAXLEN1];
        int col, n;

        /* get search mask into pat */
        fmtsout("search mask ?  ",0);
        getcmnd(pat,15);
        fmtcrlf();

        if (pat[0] == EOS) {
                return -1;      /* bug fix */
        }

        /* search all lines between from and to for pat */
        while (from <= to) {
                if (chkkey() == YES) {
                        break;
                }
                bufgo(from++);
                if (bufatbot() == YES) {
                        break;
                }
                n=bufgetln(line,MAXLEN);
                n=min(n,MAXLEN);
                line[n]=EOS;

                /* ^ anchors search */
                if (pat[0] == '^') {
                        if (amatch(line,pat+1,0) == YES) {
                                if (flag == NO) {
                                        fmtcrlf();
                                        putdec(bufln(),5);
                                        fmtsout(line,5);
                                        outdeol();
                                }
                                else {
                                        return 0;
                                }
                        }

                        continue;
                }

                /* search whole line for match */
                col=0;
                while (col < n) {
                        if (amatch(line,pat,col++) == YES) {
                                if (flag == NO) {
                                        fmtcrlf();
                                        putdec(bufln(),5);
                                        fmtsout(line,5);
                                        outdeol();
                                        break;
                                }
                                else {
                                        return col-1;
                                }
                        }
                }
        }

        /* all searching is finished */
        if (flag == YES) {
                return -1;
        }
        else {
                fmtcrlf();
        }
}

/* set tab stops for fmt routines */

tabs(args)
char *args;
{
        int n, junk;

        if (get2args(args,&n,&junk) == ERR) {
                return;
        }
        fmtset(n);
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
}

/* return YES if buffer may be drastically changed */

chkbuf()
{
        if (bufchngd == NO) {

                /* buffer not changed, no problem */
                return YES;
        }

        fmtsout("buffer not saved, proceed ?  ",0);
        pmtline();

        if (tolower(syscout(syscin())) != 'y') {
                fmterlf();
                message("cancelled");
                return NO;
        }
        else {
                fmterlf();
                return YES;
        }


}
/* print message from a command */

message(s)
char *s;
{
        fmtsout(s,0);
        fmterlf();
}
/* get two arguments the argument line args.
 * no arguments imply 1 HUGE.
 * one argument implies both args the same.
 */
get2args(args,val1,val2)
char *args;
int *val1, *val2;
{
        /* skip over the command */
        args=skiparg(args);
        args=skipbl(args);

        if (*args == EOS) {
                *val1=1;
                *val2=HUGE;
                return OK;
        }
        /* check first argument */
        if (number(args,val1) == NO) {
                message("bad argument");
                return ERR;
        }

        /* skip over first argument */
        args=skiparg(args);
        args=skipbl(args);

        /* 1 arg: arg 2 is HUGE */
        if (*args == EOS) {
                *val2=HUGE;
                return OK;
        }

        /* check second argument */
        if (number(args,val2) == NO) {
                message("bad argument");
                return ERR;
        }
        else {
                return OK;
        }
}

/* Get exactly three arguments. */

get3args(args, val1, val2, val3)
char *args;
int *val1, *val2, *val3;
{
        /* Skip the command. */
        args = skiparg (args);
        args = skipbl (args);

        /* Check first arg. */
        if(*args == EOS) {
                message("missing arguments");
                return ERR;
        }

        if (number (args, val1) == NO) {
                message("bad argument");
                return ERR;
        }

        /* Skip over first argument. */
        args = skiparg(args);
        args = skipbl(args);

        /* Check second argument. */
        if (*args == EOS) {
                message("missing arguments");
                return ERR;
        }

        if (number(args, val2) == NO) {
                message("bad argument");
                return ERR;
        }

        /* Skip over third argument. */
        args = skiparg(args);
        args = skipbl(args);

        /* Check third argument. */
        if (*args == EOS) {
                message("missing arguments");
                return ERR;
        }

        if (number (args, val3) == NO) {
                message("bad argument");
                return ERR;
        }
        else {
                return OK;
        }
}


/* skip over all except EOS, and blanks */

skiparg(args) char *args;
{
        while ( (*args != EOS) & (*args!=' ') ) {
                args++;
        }
        return args;
}


/* skip over all blanks */

skipbl(args) char *args;
{
        while (*args == ' ') {
                args++;

        }
        return args;
}


/* return YES if the user has pressed any key.
 * blanks cause a transparent pause.
 */

chkkey()
{
        int c;

        c=syscstat();
        if (c == -1) {          /* bug fix */

                /* no character at keyboard */
                return NO;
        }
        else if (c == ' ') {

                /* pause.  another blank ends pause */
                pmtline();
                if (syscin() == ' ') {
                        return NO;
                }
        }

        /* we got a nonblank character */
        return YES;
}


/* anchored search for pattern in text line at column col.
 * return YES if the pattern starts at col.
 */

amatch(line,pat,col)
char *line, *pat;
int col;
{
        int k;

        k=0;
        while (pat [k] != EOS) {
                if (pat [k] == line[col]) {
                        k++;
                        col++;
                }
                else if ((pat [k] == '?')&(line[col] != EOS)) {

                        /* question mark matches any char */
                        k++;
                        col++;
                }
                else {
                        return NO;
                }
        }
```

```
        /* the entire pattern matches */
        return YES;
}


/* replace oldpat in oldline by newpat starting at col.


 * put result in newline.
 * return number of characters in newline.
 */

replace(oldline,newline,oldpat,newpat,col)
char *oldline, *newline, *oldpat, *newpat;
int col;
{
        int k;
        char *tail, *pat;

        /* copy oldline preceding col to newline */
        k=0;
        while (k < col) {
                newline [k++]=*oldline++;
        }

        /* remember where end of oldpat in oldline is */
        tail=oldline;
        pat=oldpat;
        while (*pat++ != EOS) {
                tail++;
        }

        /* copy newpat to newline.
         * use oldline and oldpat to resolve question marks
         * in newpat.
         */
        while (*newpat != EOS) {
                if (k > MAXLEN-1) {
                        message("new line too long");
                        return ERR;
                }
                if (*newpat != '?') {
                        /* copy newpat to newline */
                        newline [k++]=*newpat++;
                        continue;
                }

                /* scan for '?' in oldpat */
                while (*oldpat != '?') {
                        if (*oldpat == EOS) {
                                message(
                                "too many ?'s in change mask"
                                );
                                return ERR;
                        }
                        oldpat++;
                        oldline++;
                }

                /* copy char from oldline to newline */
                newline [k++]=*oldline++;
                oldpat++;
                newpat++;
        }

        /* copy oldline after oldpat to newline */
        while (*tail != EOS) {
                if (k >= MAXLEN-1) {
                        message("new line too long");
                        return ERR;
                }
                newline [k++]=*tail++;
        }
        newline [k]=EOS;

        return k;
}
```

```
/*
        RED window module -- small-C version

        Source:  red%.sc
        Version: January 23, 1983

        Copyright (C) 1983 by Edward K. Ream
*/

/* data global to this module */

char    editbuf[MAXLEN];        /* the edit buffer      */
int     editp;                  /* cursor: buffer index */
int     editpmax;               /* length of buffer     */
int     edcflag;                /* buffer change flag   */

/* abort any changes made to current line */
edabt()
{
        /* get unchanged line and reset cursor */
        edgetln();
        edredraw();
        edbegin();
        edcflag = NO;
}

/* put cursor at beginning of current line */

edbegin()
{
        editp = 0;
        outxy(0,outgety());
}

/* change editbuf[editp] to c
 * don't make change if line would become to long
 */

edchng(c) char c;
{
        char oldc;
        int k;
        /* if at right margin then insert char */
        if (editp >= editpmax) {
                edinsc(c);
                return;
        }
        /* change char and print length of line */
        oldc = editbuf[editp];
        editbuf[editp] = c;
        fmtadj(editbuf,editp,editpmax);
        k = fmtlen(editbuf,editp+1);
        if (k > SCRNW-1) {

                /* line would become too long */
                /* undo the change */
                editbuf[editp] = oldc;
                fmtadj(editbuf,editp,editpmax);
        }
        else {

                /* set change flag, redraw line */
                edcflag = YES;
                editp++;
                edadj();
                edredraw();
        }
}

/* delete the char to left of cursor if it exists */

eddel()
{
        int k;

        /* just move left one column if past end of line */
        if (edxpos() < outgetx()) {
                outxy(outgetx()-1, outgety());
                return;
        }

        /* do nothing if cursor is at left margin */
        if (editp == 0) {
                return;
        }
        edcflag = YES;
        /* compress buffer (delete char) */
        k = editp;
        while (k < editpmax) {
                editbuf[k-1] = editbuf[k];
                k++;
        }

        /* update pointers, redraw line */
        editp--;
        editpmax--;
        edredraw();
}

/* edit the next line.  do not go to end of buffer */
eddn()
{
        int oldx;

        /* save visual position of cursor */
        oldx = outgetx();

        /* replace current edit line */
        edrepl();

        /* do not go past last non-null line */
        if (bufnrbot()) {
                return;
        }

        /* move down one line in buffer */
        bufdn();
        edgetln();
        /* put cursor as close as possible on this
         * new line to where it was on the old line.
         */
        editp = edscan(oldx);

        /* update screen */
        if (edatbot()) {
                edsup(bufln()-SCRNL2);
                outxy(oldx, SCRNL1);
        }
        else {
                outxy(oldx, outgety()+1);
        }
        return;
}

/* put cursor at the end of the current line */

edend()
{
        editp = editpmax;
        edadj();
        outxy(edxpos(),outgety());
}


/* start editing line c
 * redraw the screen with cursor at position p
 */
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
edgo(n, p) int n, p;
{
        /* replace current line */
        edrepl();

        /* go to new line */
        bufgo(n);

        /* prevent going past end of buffer */
        if (bufatbot()) {
                bufup();
        }

        /* redraw the screen */
        bufout(bufln(),1,SCRNL1);
        edgetln();
        editp = min(p, editpmax);
        outxy(edxpos(), 1);
        return;
}


/* insert c into the buffer if possible */

edins(c)
char c;
{
        int k;

        /* do nothing if edit buffer is full */
        if (editpmax >= MAXLEN) {
                return;
        }

        /* fill out line if we are past its end */
        if ((editp == editpmax) & (edxpos() < outgetx())) {
                k = outgetx() - edxpos();
                editpmax = editpmax + k;
                while (k-- > 0) {

                        edittbuf [editp++] = ' ';
                }
                editp = editpmax;
        }

        /* make room for inserted character */
        k = editpmax;
        while (k > editp) {
                edittbuf[k] = edittbuf[k-1];
                k--;
        }

        /* insert character, update pointers */
        edittbuf[editp] = c;
        editp++;
        editpmax++;

        /* recalculate print length of line  */
        fmtadj(edittbuf,editp-1,editpmax);
        k = fmtlen(edittbuf,editp);
        if ( (k > SCRNW1) & (editp == editpmax) ) {
                /* auto-split the line (line wrap) */

                /* scan for the start of the current word */
                k = editp - 1;
                while ( (k >= 0) &
                        (edittbuf[k] != ' ') &
                        (edittbuf[k] != TAB)
                      ) {
                        k--;
                }

                /* never split a word */
                if (k < 0) {
                        eddel();
                        return;
                }

                /* split the line at the current word */
                editp = k + 1;
                edsplit();
                edend();
        }
        else if (k > SCRNW1) {

                /* line would become too long */
                /* delete what we just inserted */
                eddel();
        }
        else {
                /* set change flag, redraw line */
                edcflag = YES;
                edredraw();
        }
}


/* join (concatenate) the current line with the one above it */

edjoin()
{
        int k, k1, k2;

        /* do nothing if at top of file */
        if (bufattop()) {
                return;

        }

        /* replace lower line temporarily */
        edrepl();

        /* get upper line into buffer */
        bufup();
        k1 = bufgetln(edittbuf, MAXLEN);

        /* append lower line to buffer */
        bufdn();
        k2 = bufgetln(edittbuf+k1, MAXLEN-k1);

        /* abort if the screen isn't wide enough */
        if (k1 + k2 > SCRNW1) {

                /* bug fix */
                bufgetln(edittbuf,MAXLEN);
                return;
        }

        /* replace upper line */
        bufup();
        editpmax = k1 + k2;
        editp = k1 + editp;
        edadj();
        edcflag = YES;
        edrepl();

        /* delete the lower line */
        bufdn();
        bufdel();
        bufup();

        /* update the screen */
        if (edattop()) {
                edredraw();
        }
        else {
                k = outgety() - 1;
                bufout(bufln(),k,SCRNL-k);
                outxy(0,k);
                edredraw();
        }
}


/* delete chars until end of line or c found */

edkill(c) char c;
{

        int k,p;

        /* do nothing if at right margin */
        if (editp == editpmax) {
                return;
        }
        edcflag = YES;

        /* count number of deleted chars */
        k = 1;
        while ((editp+k) < editpmax) {
                if (edittbuf[editp+k] == c) {
                        break;
                }
                else {
                        k++;
                }
        }

        /* compress buffer (delete chars) */
        p = editp+k;
        while (p < editpmax) {
                edittbuf[p-k] = edittbuf[p];
                p++;
        }

        /* update buffer size, redraw line */
        editpmax = editpmax-k;
        edredraw();
}

/* move cursor left one column.
 * never move the cursor off the current line.
 */

edleft()
{
int k;

        /* if past right margin, move left one column */
        if (edxpos() < outgetx()) {
                outxy(max(0, outgetx()-1), outgety());
        }

        /* inside the line,  move left one character */
        else if (editp != 0) {
                editp--;
                outxy(edxpos(),outgety());
        }
}


/* insert a new blank line below the current line */

ednewdn()
{
        int k;

        /* make sure there is a current line and
         * put the current line back into the buffer.
         */
```

```
        if (bufatbot()) {
                bufins(editbuf,editpmax);
        }
        edrepl();

        /* move past current line */
        bufdn();

        /* insert place holder:  zero length line */
        bufins(editbuf,0);

        /* start editing the zero length line */
        edgetln();

        /* update the screen */
        if (edatbot()) {

                /* note: bufln()  >= SCRNL */
                edsup(bufln()-SCRNL2);
                outxy(edxpos(),SCRNL1);




        }
        else {
                k = outgety();
                bufout(bufln(),k+1,SCRNL1-k);
                outxy(edxpos(),k+1);
        }
}


/* insert a new blank line above the current line */

ednewup()
{
int k;
        /* put current line back in buffer */
        edrepl();

        /* insert zero length line at current line */
        bufins(editbuf,0);

        /* start editing the zero length line */
        edgetln();

        /* update the screen */
        if (edattop()) {
                edsdn(bufln());
                outxy(edxpos(),1);
        }
        else {
                k = outgety();
                bufout(bufln(),k,SCRNL-k);
                outxy(edxpos(),k);
        }
}


/* move cursor right one character.
 * never move the cursor off the current line.
 */

edright()

{
        /* if we are outside the line move right one column */
        if (edxpos() < outgetx()) {
                outxy (min(SCRNW1, outgetx()+1), outgety());
        }

        /* if we are inside a tab move to the end of it */
        else if (edxpos() > outgetx()) {
                outxy (edxpos(), outgety());
        }

        /* move right one character if inside line */
        else if (editp < editpmax) {
                editp++;
                edadj();
                outxy(edxpos(),outgety());
        }

        /* else move past end of line */
        else {
                outxy (min(SCRNW1, outgetx()+1), outgety());
        }
}

/* split the current line into two parts.
 * scroll the first half of the old line up.
 */

edsplit()
{
        int p, q;
        int k;

        /* indicate that edit buffer has been saved */
        edcflag = NO;

        /* replace current line by the first half of line */
        if (bufatbot()) {
                bufins(editbuf, editp);
        }
        else {
                bufrepl(editbuf, editp);
        }

        /* redraw the first half of the line */
        p = editpmax;
        q = editp;
        editpmax = editp;
        editp = 0;
        edredraw();

        /* move the second half of the line down */
        editp = 0;
        while (q < p) {
                editbuf [editp++] = editbuf [q++];
        }
        editpmax = editp;
        editp = 0;

        /* insert second half of the line below the first */
        bufdn();
        bufins(editbuf, editpmax);

        /* scroll the screen up and draw the second half */
        if (edatbot()) {
                edsup(bufln()-SCRNL2);
                outxy(1,SCRNL1);
                edredraw();
        }
        else {
                k = outgety();
                bufout(bufln(), k+1, SCRNL1-k);
                outxy(1, k+1);
                edredraw();
        }
}


/* move cursor right until end of line or
 * character c found.
 */

edsrch(c) char c;
{
        /* do nothing if at right margin */
        if (editp == editpmax) {
                return;
        }

        /* scan for search character */

        editp++;
        while (editp < editpmax) {
                if (editbuf[editp] == c) {
                        break;
                }
                else {
                        editp++;
                }
        }

        /* reset cursor */
        edadj();
        outxy(edxpos(),outgety());
}


/* move cursor up one line if possible */

edup()
{
        int oldx;

        /* save visual position of cursor */
        oldx = outgetx();

        /* put current line back in buffer */
        edrepl();

        /* done if at top of buffer */
        if (bufattop()) {
                return;
        }

        /* start editing the previous line */
        bufup();
        edgetln();

        /* put cursor on this new line as close as
         * possible to where it was on the old line.
         */
        editp = edscan(oldx);

        /* update screen */
        if (edattop()) {
                edsdn(bufln());
                outxy(oldx, 1);
        }
        else {
                outxy(oldx, outgety()-1);
        }
        return;
}

/* delete the current line */

edzap()
{
        int k;

        /* delete the line in the buffer */
        bufdel();

        /* move up one line if now at bottom */
        if (bufatbot()) {
```

(Continued on page 82)

RED - Listing

(Listing continued, text begins on page 62)

```
                bufup();
                edgetln();
                /* update screen */
                if (edattop()) {
                        edredraw();
                }
                else {
                        outdelln();
                        outxy(0,outgety()-1);
                }
                return;
        }

        /* start editing new line */
        edgetln();

        /* update screen */
        if (edattop()) {
                edsupi(buflin());
                outxy(0,1);
        }
        else {
                k = outgety();
                bufout(buflin(),k,SCRNL-k);
                outxy(0,k);
        }
}


/* ----- utility routines (not used outside this file) ----- */


/* adjust the cursor so it stays on the screen.
 * call this routine whenever the cursor could move right.
 */

edadj()
{
        while (fmtlen(editbuf, editp) > SCRNW1) {
                editp--;
        }
}


/* return true if the current edit line is being
 * displayed on the bottom line of the screen.
 */

edatbot()
{
        return outgety() == SCRNL1;
}


/* return true if the current edit line is being
 * displayed on the bottom line of the screen.
 */

edattop()
{
        return outgety() == 1;
}


/* redraw edit line from index to end of line */
/* reposition cursor */
edredraw()
{
        fmtadj(editbuf,0,editpmax);
        fmtsubs(editbuf,max(0,editp-1),editpmax);
        outxy(edxpos(),outgety());
}


/* return the x position of the cursor on screen */

edxpos()
{
        return fmtlen(editbuf, editp);
}


/* fill edit buffer from current main buffer line.
 * the caller must check to make sure the main
 * buffer is available.
 */

edgetln()
{
        int k;

        /* put cursor on left margin, reset flag */
        editp = 0;
        edcflag = NO;

        /* get edit line from main buffer */
        k = bufgetln(editbuf,MAXLEN);
        if (k > MAXLEN) {
                error("line truncated");
                editpmax = MAXLEN;
        }
        else {
                editpmax = k;
        }
        fmtadj(editbuf,0,editpmax);
}

/* Replace current main buffer line by edit buffer.
 * The edit buffer is NOT changed or cleared.
 */

edrepl()
{
        /* do nothing if nothing has changed */
        if (edcflag == NO) {
                return;
        }

        /* make sure we don't replace the line twice */
        edcflag = NO;

        /* insert instead of replace if at bottom of file */
        if (bufatbot()) {
                bufins(editbuf,editpmax);
        }
        else {
                bufrepl(editbuf,editpmax);
        }
}


/* Set editp to the largest index such that
 * buf[editp] will be printed <= xpos
 */

edscan(xpos) int xpos;
{
        editp = 0;
        while (editp < editpmax) {
                if (fmtlen(editbuf,editp) < xpos) {
                        editp++;
                }
                else {
                        break;
                }
        }
        return editp;
}


/* scroll the screen up.  topline will be new top line */

edsup(topline) int topline;
{
        if (outhasup() == YES) {

                /* hardware scroll */
                outsup();

                /* redraw bottom line */
                bufout(topline+SCRNL2,SCRNL1,1);
        }
        else {

                /* redraw whole screen */
                bufout(topline,1,SCRNL1);
        }
}
/* scroll screen down.  topline will be new top line */

edsdn(topline) int topline;
{
        if (outhasdn() == YES) {

                /* hardware scroll */
                outsdn();

                /* redraw top line */
                bufout(topline,1,1);
        }
        else {

                /* redraw whole screen */
                bufout(topline,1,SCRNL1);
        }
}
```

```
/*
        RED output format module -- small-C version

        Source:  red5.sc
        Version: August 21, 1982.

        Copyright (C) 1983 by Edward K. Ream
*/


/* Define variables global to this module. */

/* define maximal length of a tab character */

int fmttab;


/* define the current device and device width */

int fmtdev;             /* device -- YES/NO = LIST/CONSOLE */
int fmtwidth;           /* device width.  LISTW/SCRNW1 */


/*
     fmtcol[i] is the first column at which
     buf[i] will be printed.
     fmtsub() and fmtlen() assume fmtcol[] is valid on entry.
*/

int fmtcol[MAXLEN+1];


/*
        Direct output from this module to either the console or
        the list device.
*/

fmtesn(listflag) int listflag;
```

```
{
        if (listflag==YES) {
                fmtdev=YES;
                fmtwidth=LISTW;
        }
        else {
                fmtdev=NO;
                fmtwidth=SCRNW1;
        }
}


/*
        Adjust fmtcol[] to prepare for calls on
        fmtout() and fmtlen().

        NOTE:  this routine is needed as an efficiency
               measure.  Without fmtadj(), calls on
               fmtlen() become too slow.
*/

fmtadj(buf,minind,maxind) char *buf; int minind,maxind;
{
int k;
        /* line always starts at left margin */
        fmtcol[0]=0;
        /* start scanning at minind */
        k=minind;
        while (k<=maxind) {

                /* comment out -----
                if (buf[k]==CR) {
                        break;
                }
                ----- end comment out */

                fmtcol[k+1]=fmtcol[k]+fmtlench(buf[k],fmtcol[k]);
                k++;
        }
}
/* return column at which at which buf[i] will be printed */

fmtlen(buf,i) char *buf; int i;
{
        return(fmtcol[i]);
}


/*
        Print buf[i] ... buf[j-1] on current device so long as
        characters will not be printed in last column.
*/

fmtsubs(buf,i,j) char *buf; int i, j;
{
int k;
        if (fmtcol[i]>=fmtwidth) {
                return;
        }
        outxy(fmtcol[i],outgety());     /* position cursor */
        while (i<j) {

                /* Comment out -----
                if (buf[i]==CR) {
                        break;
                }
                ----- end comment out */

                if (fmtcol[i+1]>fmtwidth) {
                        break;
                }
                fmtoutch(buf[i],fmtcol[i]);
                i++;
        }
        outdeol();      /* clear rest of line */
}
/*
    Print string which ends with CR or EOS to current device.
    Truncate the string if it is too long.
*/

fmtout(buf,offset) char *buf; int offset;
{
char c;
int col,k;
        col=0;
        while (c=*buf++) {
                if (c==CR) {
                        break;
                }
                k=fmtlench(c,col);
                if ((col+k+offset)>fmtwidth) {
                        break;
                }
                fmtoutch(c,col);
                col=col+k;
        }
}
/* Return length of char c at column col. */

fmtlench(c,col) char c; int col;
{
        if (c==TAB) {
                /* tab every fmttab columns */
                return(fmttab-(col%fmttab));
        }
        else if (c<32) {
                /* control char */
                return(2);
        }
        else {
                return(1);
        }
}


/*
        Output one character to current device.
        Convert tabs to blanks.
*/

fmtoutch(c,col) char c; int col;
{
int k;
        if (c==TAB) {
                k=fmtlench(TAB,col);
                while ((k--)>0) {
                        fmtdevch(' ');
                }
        }
        else if (c<32) {
                fmtdevch('^');
                fmtdevch(c+64);
        }
        else {
                fmtdevch(c);
        }
}


/* Output character to current device. */

fmtdevch(c) char c;
{
        if (fmtdev==YES) {
                sysout(c & 127);
        }
        else {
                outchar(c & 127);
        }
}


/* Output a CR and LF to the current device. */

fmtcrlf()
{
        if (fmtdev==YES) {
                sysout(CR);
                sysout(LF);
        }
        else {
                /* kludge: this should be in out module */
                /* make sure out module knows position */
                outxy(0,SCRNL1);
                syscout(CR);
                syscout(LF);
        }
}


/* Set tabs at every n columns. */

fmtset(n) int n;
{
        fmttab=max(1,n);
}
```

```
/*
RED terminal output module
Source:  red6.sc
This file was created by the configuration program:
January 20, 1983; February 25, 1983

Modified by hand: gotoxy(), outdeol().
*/

/*
Define the current coordinates of the cursor.
*/

int outx, outy;


/*
Return the current coordinates of the cursor.
*/

outgetx()
{
        return(outx);
}

outgety()
{
        return(outy);
}

/*
Output one printable character to the screen.
*/

outchar(c) char c;
{
        syscout(c);
        outx++;
        return(c);
}

/*
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
Position cursor to position x,y on screen.
0,0 is the top left corner.
*/
outxy(x,y) int x,y;
{
                syscout(11);
                syscout(64+y);
                syscout(16);
                syscout( ((x/10)<<4) | (x%10) );
        outx=x;
        outy=y;
}

/*
Erase the entire screen.
Make sure the rightmost column is erased.
*/
outclr()
{
        int k;

        k=0;
        while (k<$CRHL) {
                outxy(0,k++);
                outdelln();
        }
        outxy(0,0);
}

/*
Delete the line on which the cursor rests.
Leave the cursor at the left margin.
*/

outdelln()
{
        outxy(0,outy);
        outdeol();
}

/*
Delete to end of line.
Assume the last column is blank.
*/

outdeol()
{
        syscout(27);
        syscout('K');
}

/*
Return yes if terminal has indicated hardware scroll.
*/

outhasup()
{
        return(YES);
}

outhasdn()
{
        return(YES);
}

/*
Scroll the screen up.
Assume the cursor is on the bottom line.
*/

outsup()
{
        /* auto scroll */
        outxy(0,$CRHL-1);
        syscout(10);
}

/*
Scroll screen down.
Assume the cursor is on the top line.
*/

outsdn()
{
        int wait;

        /* auto scroll */
        outxy(0,0);
        syscout(27);
        syscout('I');

        /* 19.2 Xmodem kludge */
        wait = 0;
        while (wait++ < 1500)
                ;
}
```

```
/*
        RED prompt line module -- small-C version

        Source:  red7.sc
        Version: August 24, 1982.

        Copyright (C) 1983 by Edward K. Ream
*/

/* Define the prompt line data. */

char pmtln[MAXLEN];     /* mode */
char pmtfn[SYSFNMAX];   /* file name */

/* Initialize the mode and file name. */

pmtclr()
{
        pmtln[0] = 0;
        pmtfn[0] = 0;
}


/*
        Put error message on prompt line.
        Wait for response.
*/

pmtmesg(s1,s2)
char *s1, *s2;
{
        int x,y;

        /* save cursor */
        x=outgetx();
        y=outgety();
        outxy(0,0);
        /* make sure line is correct */
        outdelln();
        pmtline1();
        pmtcol1(x);
        /* output error message */
        fmtsout(s1,outgetx());
        fmtsout(s2,outgetx());
        /* wait for input from console */
        syscin();
        /* redraw prompt line */
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}


/* Write new mode message on prompt line. */

pmtmode(s)
char *s;
{
        int x,y;                /* save cursor on entry */

        /* save cursor */
        x=outgetx();
        y=outgety();

        /* redraw whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(s);
        /* restore cursor */
        outxy(x,y);
}


/* Update file name on prompt line. */

pmtfile(s)
char *s;
{
        int x, y;

        /* save cursor */
        x=outgetx();
        y=outgety();
        /* update whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1(x);             /* bug fix -- 1/28/82 */
        pmtfile1(s);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}


/* Change mode on prompt line to edit: */

pmtedit()
{
        pmtmode("edit:");
}


/* Update line and column numbers on prompt line. */

pmtline()
{
        int x,y;

        /* save cursor */
        x=outgetx();
        y=outgety();
        /* redraw whole line */
        outxy(0,0);
        outdelln();
        pmtline1();
        pmtcol1(x);
        pmtfile1(pmtfn);
        pmtmode1(pmtln);
        /* restore cursor */
        outxy(x,y);
}
```

```
/* Update just the column number on prompt line. */

pmtcol()
{
        int x,y;

        /* save cursor */
        x=outgetx();
        y=outgety();
        /* update column number */
        pmtcol1(x);
        /* update cursor */
        outxy(x,y);
}


/* Update mode.  call getcmnd() to write on prompt line. */

pmtcmnd(mode,buffer)
char *mode, *buffer;
{
        int x,y;

        /* save cursor */
        x=outgetx();
        y=outgety();
        pmtmode1(mode);
        /* user types command on prompt line */
        getcmnd(buffer,outgetx());
        /* restore cursor */

        /* --- new ---
        outxy(x,y);
        ----- end comment out */
}


/* Update and print mode. */

pmtmode1(s)
char *s;
{
        int i;

        outxy(40,0);
        fmtsout(s,40);
        i=0;
        while (pmtln[i++]=*s++) {
                ;
        }
}


/* Print the file name on the prompt line. */

pmtfile1(s)
char *s;
{
        int i;

        outxy(25,0);
        if (*s==EOS) {
                fmtsout("no file",25);
        }
        else {
                fmtsout(s,25);
        }
        i=0;
        while (pmtfn[i++]=*s++) {
                ;
        }
}


/* Print the line number on the prompt line. */

pmtline1()
{
        outxy(0,0);
        fmtsout("line: ",0);
        putdec(bufln(),5);
}

/* Print column number of the cursor. */

pmtcol1(x)
int x;
{

        /* comment out all the following code if you do not
         * want column numbers to be drawn on the screen.
         * some people complain of too much flicker.
         */

        outxy(12,0);
        fmtsout("column: ",12);
        putdec(x,3);
}
```

```
/*
        RED general utilities -- small-C version

        Source:  red9.sc
        Version: August 27, 1982.

        Copyright (C) 1983 by Edward K. Ream
*/
```

```
/*      Return larger of two numbers.   */

max(a, b)
int a, b;
{
        if (a >= b) {
                return a;
        }
        else {
                return b;
        }
}


/*      Return smaller of two numbers.  */

min(a, b)
int a, b;
{
        if (a <= b) {
                return a;
        }
        else {
                return b;
        }
}


/*      Return the absolute value of a number.  */

abs(n)
int n;
{
        if (n < 0) {
                return -n;
        }
        else {
                return n;
        }
}


/*      Convert a character to lower case.      */

tolower(c)
char c;
{
        if ( (c >= 'A') & (c <= 'Z') ) {
                return c - 'A' + 'a';
        }
        else {
                return c;
        }
}

/*
        return: is first token in args a number ?
        return value of number in *val
*/

number(args,val) char *args; int *val;
{
char c;
        c=*args++;
        if ((c<'0')|(c>'9')) {
                return(NO);
        }
        *val=c-'0';
        while (c=*args++) {
                if ((c<'0')|(c>'9')) {
                        break;
                }
                *val=(*val*10)+c-'0';
        }
        return(YES);
}


/* Convert character buffer to numeric. */

ctoi(buf,index) char *buf; int index;
{
int k;
        while ( (buf[index]==' ') ||
                (buf[index]==TAB) ) {
                index++;
        }
        k=0;
        while ((buf[index]>='0')&(buf[index]<='9')) {
                k=(k*10)+buf[index]-'0';
                index++;
        }
        return(k);
}


/*
        Put decimal integer n in field width >= w.
        Left justify the number in the field.
*/

putdec(n,w) int n,w;
{
char chars[10];
int i,nd;
        nd=itoc(n,chars,10);
        i=0;
        while (i<nd) {
                sysout(chars[i++]);
        }


        i=nd;
        while (i++<w) {
                sysout(' ');
        }
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
 * Convert integer n to character string in str. */

itoa(n,str,size) int n; char *str; int size;
{
int absval;
int len;
int i,j,k;
        absval=abs(n);
        /* generate digits */
        str[0]=0;
        i=1;
        while (i<size) {
                str[i++]=(absval%10)+'0';
                absval=absval/10;
                if (absval==0) {
                        break;
                }
        }
        /* generate sign */
        if ((i<size)&(n<0)) {
                str[i++]='-';
        }
        len=i-1;
        /* reverse sign, digits */
        i--;
        j=0;
        while (j<i) {
                k=str[i];
                str[i]=str[j];
                str[j]=k;
                i--;
                j++;
        }
        return(len);
}


/* System error routine. */

syserr(s) char *s;
{
        pmtmesg("system error: ",s);
}


/* User error routine. */

error(s) char *s;
{
        pmtmesg("error: ",s);
}
```

End Listing One

Listing Two

```
#asm
; Part I of the Small-C Run-time Library
; Functions called from the code generators
;
; Source:  lib1
; Version: August 5, 1982
;
;
;Fetch a single byte from the address in HL and
; sign extend into HL
CCCHAR: MOV     A,M
CCSXT:  MOV     L,A
        RLC
        SBB     A
        MOV     H,A
        RET
;Fetch a full 16-bit integer from the address in HL
CCINT:  MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A
        RET
;Store a single byte from HL at the address in DE
CCPCHAR: MOV    A,L
        STAX    D
        RET
;Store a 16-bit integer in HL at the address in DE
CCPINT: MOV     A,L
        STAX    D
        INX     D
        MOV     A,H
        STAX    D
        RET
;Inclusive "or" HL and DE into HL
CCOR:   MOV     A,L
        ORA     E
        MOV     L,A
        MOV     A,H
        ORA     D
        MOV     H,A
        RET
;Exclusive "or" HL and DE into HL
CCXOR:  MOV     A,L
        XRA     E
        MOV     L,A
        MOV     A,H
        XRA     D
        MOV     H,A
        RET
;"And" HL and DE into HL
CCAND:  MOV     A,L
        ANA     E
        MOV     L,A
        MOV     A,H
        ANA     D
        MOV     H,A
        RET
;Test if HL = DE and set HL = 1 if true else 0
CCEQ:   CALL    CCCMP
        RZ
        DCX     H
        RET
;Test if DE != HL
CCNE:   CALL    CCCMP
        RNZ
        DCX     H
        RET
;Test if DE > HL (signed)
CCGT:   XCHG
        CALL    CCCMP
        RC
        DCX     H
        RET
;Test if DE <= HL (signed)
CCLE:   CALL    CCCMP
        RZ
        RC
        DCX     H
        RET
;Test if DE >= HL (signed)
CCGE:   CALL    CCCMP
        RNC
        DCX     H
        RET
;Test if DE < HL (signed)
CCLT:   CALL    CCCMP
        RC
        DCX     H
        RET
;Common routine to perform a signed compare
; of DE and HL
;This routine performs DE - HL and sets the conditions:
;       Carry reflects sign of difference (set means DE < HL)
;       Zero/non-zero set according to equality.
CCCMP:  MOV     A,E
        SUB     L
        MOV     E,A
        MOV     A,D
        SBB     H
        LXI     H,1     ;preset true condition
        JM      CCCMP1
        ORA     E       ;"OR" resets carry
        RET
CCCMP1: ORA     E
        STC             ;set carry to signal minus
        RET
;
;Test if DE >= HL (unsigned)
CCUGE:  CALL    CCUCMP
        RNC
        DCX     H
        RET
;
;Test if DE < HL (unsigned)
CCULT:  CALL    CCUCMP
        RC
        DCX     H
        RET
;
;Test if DE > HL (unsigned)
CCUGT:  XCHG
        CALL    CCUCMP
        RC
        DCX     H
        RET
;
;Test if DE <= HL (unsigned)
CCULE:  CALL    CCUCMP
        RZ
        RC
        DCX     H
        RET
;
;Common routine to perform unsigned compare
;carry set if DE < HL
;zero/nonzero set accordingly
CCUCMP: MOV     A,D
        CMP     H
        JNZ     $+5
        MOV     A,E
        CMP     L
        LXI     H,1
        RET
;
;Shift DE arithmetically right by HL and return in HL
CCASR:  XCHG
        MOV     A,H
        RAL
        MOV     A,H
        RAR
```

```
        MOV     H,A
        MOV     A,L
        RAR
        MOV     L,A
        DCR     E
        JNZ     CCASR.1
        RET
;Shift DE arithmetically left by HL and return in HL
CCASL:  XCHG
        DAD     H
        DCR     E
        JNZ     CCASL.1
        RET
;Subtract HL from DE and return in HL
CCSUB:  MOV     A,E
        SUB     L
        MOV     L,A
        MOV     A,D
        SBB     H
        MOV     H,A
        RET
;Form two's complement of HL
;
CCNEG:  CALL    CCCOM
        INX     H
        RET
;Form the one's complement of HL
CCCOM:  MOV     A,H
        CMA
        MOV     H,A
        MOV     A,L
        CMA
        MOV     L,A
        RET

;Multiply DE by HL and return in HL
CCMULT: MOV     B,H
        MOV     C,L
        LXI     H,0
CCMULT1: MOV    A,C
        RRC
        JNC     $+4
        DAD     D
        XRA     A
        MOV     A,B
        RAR
        MOV     B,A
        MOV     A,C
        RAR
        MOV     C,A
        ORA     B
        RZ
        XRA     A
        MOV     A,E
        RAL
        MOV     E,A
        MOV     A,D
        RAL
        MOV     D,A
        ORA     E
        RZ
        JMP     CCMULT1
;Divide DE by HL and return quotient in HL, remainder in DE
CCDIV:  MOV     B,H
        MOV     C,L
        MOV     A,D
        XRA     H
        PUSH    PSW
        MOV     A,D
        ORA     A
        CM      CCDENEG
        MOV     A,H
        ORA     A
        CM      CCHLNEG
        MVI     A,16
        PUSH    PSW
        XCHG
        LXI     D,0
CCDIV1: DAD     H
        CALL    CCRDEL
        JZ      CCDIV2
        CALL    CCCMPBCDE
        JM      CCDIV2
        MOV     A,L
        ORI     1
        MOV     L,A
        MOV     A,E
        SUB     C
        MOV     E,A
        MOV     A,D
        SBB     B
        MOV     D,A
CCDIV2: POP     PSW
        DCR     A
        JZ      CCDIV3
        PUSH    PSW
        JMP     CCDIV1
CCDIV3: POP     PSW
        RP
        CALL    CCDENEG
        XCHG
        CALL    CCDENEG
        XCHG
        RET
CCDENEG: MOV    A,D
        CMA
        MOV     D,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
        RET

CCHLNEG: MOV    A,B
        CMA
        MOV     B,A
        MOV     A,C
        CMA
        MOV     C,A
        INX     B
        RET
CCRDEL: MOV     A,E
        RAL
        MOV     E,A
        MOV     A,D
        RAL
        MOV     D,A
        ORA     E
        RET
CCCMPBCDE: MOV  A,E
        SUB     C
        MOV     A,D
        SBB     B
        RET
#endasm
```

```
#asm
; library for small-C -- Part II
;
; Source:  lib2
; Version: September 3, 1982
;
; Adapted from: CCC.ASM (C.CCC).  BDS C version 1.45,  11/22/81
; Original written by Leor Zolman
; Modification by Edward K. Ream
;


base:   equ     0       ;start of ram in system
bdos:   equ     base+5
tpa:    equ     base+100h
fcb:    equ     base+5ch
nfcbs:  equ     8       ;maximum # of files open at one time
tbuff:  equ     base+80h
origin: equ     tpa
exitad: equ     base    ;warm boot location

;
; define ASCII codes;
;

cr:     equ 0dh         ;carriage return
lf:     equ 0ah         ;linefeed
newlin: equ lf          ;newline
tab:    equ 9           ;tab
bs:     equ 08h         ;backspace
ctrlc:  equ 3           ;control-C

;
; define BDOS call codes:
;

 errorv  equ 255        ;error value from BDOS

 conin:  equ 1          ;get a character from console
 conout: equ 2          ;write a character to console
 lstout: equ 5          ;write a character to list device
 dconio: equ 6          ;direct console I/O (only for CP/M 2.0)
 pstrng: equ 9          ;print string (terminated by '$')
 getlin: equ 10         ;get buffered line from console
 cstat:  equ 11         ;get console status
 select: equ 14         ;select disk
 open:   equ 15         ;open a file
 closec: equ 16         ;close a file
 delc:   equ 19         ;delete a file
 reads:  equ 20         ;read a sector (sequential)
 writs:  equ 21         ;write a sector (sequential)
 create: equ 22         ;make a file
 renc:   equ 23         ;rename file
 sdma:   equ 26         ;set dma
 readr:  equ 33         ;read random sector
 writr:  equ 34         ;write random sector
 ofsize: equ 35         ;compute file size
 srecc:  equ 36         ;set random record

 ;
 ; Define global variables used by the library
 ;

 args:   ds      10      ;"arghak" puts argc here.
 arg1    equ     args
 arg2    equ     args+2
 arg3    equ     args+4
 arg4    equ     args+6
 arg5    equ     args+8
 arg6    equ     args+10
 arg7    equ     args+12

 iohook: ds      6       ;room for I/O subroutines for use
                         ;by "inp" and "outp" library routines
                         ;(initialized by init)

 rseed   ds      8       ;seed for random number routines
                         ;(initialized by init)

 tmp:    ds      1       ; misc. garbage space
 tmp1:   ds      2       ;
 tmp2:   ds      2       ;
 tmp2a:  ds      2       ;

 ungetl: ds      1       ;where characters are "ungotten"
 lastc:  ds      1       ;last char typed

 allocp: ds      2       ;storage allocation pointers
 allocmx: ds     2       ;(initialized by init)
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
;
; The fcb table (fcbt): 36 bytes per file control block
;

fcbt:   ds      36*nfcbs        ;reserve room for fcb's

;
; The fd table: one byte per file specifying r/w/open as follows:
;       bit 0 is high if open, low if closed
;       bit 1 is high if open for read
;       bit 2 is high if open for write
; (both b1 and b2 may be high)
;

fdt:    ds      nfcbs   ;one byte per fcb

;
; The command line is copied here by init:
;

comlin: ds      131     ;copy of the command line

;
; This is where "init" places the array of argument pointers:
;
; the "argv" parameter points 2 bytes before arglst.
; thus, up to 30 parameters may be passed to main().

arglst: ds      60


;
; Because small-C pushes arguments in the order in which
; they appear in a program, it is not possible to know where
; the first argument is unless the number of arguments is
; known.
;
; The following three routines move arguments from the
; stack into the argument area, depending on the number
; of arguments to the function.
;

get3args:       lxi     h,4     ;point at arg 3
                dad     sp

                push    h       ;get it
                mov     a,m
                inx     h
                mov     h,m
                mov     l,a
                shld    arg3
                pop     h
                inx     h       ;point at arg 2
                inx     h
                jmp     get2next

get2args:       lxi     h,4     ;point at arg 2
                dad     sp

get2next:       push    h
                mov     a,m     ;get arg 2 from stack
                inx     h
                mov     h,m
                mov     l,a
                shld    arg2    ;store it
                pop     h
                inx     h
                inx     h
                jmp     get1next

get1arg:        lxi     h,4     ;point at arg 1
                dad     sp

get1next:       mov     a,m     ;get arg 1 from stack
                inx     h
                mov     h,m
                mov     l,a
                shld    arg1    ;store it
                ret


;
; These routines get the indicated parameter into both
; the a and hl regs, assuming that the correct call to
; either get1arg, get2args or get3args has been made.
;

ma1to:          lhld    arg1
                mov     a,l
                ret

ma2to:          lhld    arg2
                mov     a,l
                ret

ma3to:          lhld    arg3
                mov     a,l
                ret

;
; This routine is called first to do argc & argv processing
; and some o&ds and ends initializations:
; --- push argc and argv in reverse order from BDS C ---
;

ccgo:
init:   pop     h       ;store return address
        shld    tmp2    ; somewhere safe for the time being

;--- push later ---
;       lxi     h,arglst-2      ;set the "argv" that the
;       push    h               ; main program will get.

;
; Initialize storage allocation pointers:
;
;--- you  can  add these if you modify the small-C compiler ---
;--- so that it defines the label freram to point after the ---
;--- end of the externals                                   ---
;
;       lxi     h,freram ;get address after end of externals
;       shld    allocp  ;store at allocation pointer for sbrk()
;       lxi     h,1000  ;default safety space between stack and
;       shld    alocmx  ; highest allocatable address in memory
                        ; (for use by "sbrk".).

;
; Initialize random seed:
;
        lxi     h,59dch ;let's stick something wierd into the
        shld    rseed   ;first 16 bits of the random-number seed

;
;Initialize I/O hack locations:
;

        mvi     a,0dbh          ;"in" op, for "in xx; ret"
        sta     ihack
        mvi     a,0d3h          ;"out" op for "out xx; ret"
        sta     ihack+3
        mvi     a,0c9h          ;"ret" for above subroutines
        sta     ihack+2         ;the port number is filled in
        sta     ihack+5         ;by "inp" and "outp"


        mvi     c,11    ;interrogate console status to see if
        call    bdos    ;there is a stray character.

        ora     a       ;used to be `ani 1'... better for
        nop             ;for some CP/M like systems

        jz      initzz
        mvi     c,1     ;if input present, clear it
        call    bdos

initzz: lxi     h,tbuff         ;if arguments given, process them.
        lxi     d,comlin        ;get ready to copy command line
        mov     b,m             ;first get length of it
        inx     h
        mov     a,b
        ora     a       ;if no arguments, don't parse for argv
        jnz     initl
        lxi     d,1     ;set argc to 1 in such a case.
        jmp     i5

initl:  mov     a,m     ;ok, there are arguments. parse...
        stax    d       ;first copy command line to comlin
        inx     h
        inx     d
        dcr     b
        jnz     initl
        xra     a       ;place zero following line
        stax    d

        lxi     h,comlin        ;now compute pointers to each arg
        lxi     d,1             ;arg count
        lxi     b,arglst        ;where pointers will all go
        xra     a               ;clear "in a string" flag
        sta     tmp1
i2:     mov     a,m     ;between args...
        inx     h
        cpi     ' '
        jz      i2
        ora     a
        jz      i5      ;if null byte, done with list
        cpi     '"'
        jnz     i2a     ;quote?
        sta     tmp1    ;yes. set "in a string" flag
        jmp     i2b

i2a:    dcx     h
i2b:    mov     a,l     ;ok, HL is a pointer to the start
        stax    b       ;of an arg string. store it.
        inx     b
        mov     a,h
        stax    b
        inx     b
        inx     d       ;bump arg count
i3:     mov     a,m
        inx     h       ;pass over text of this arg
        ora     a       ;if at end, all done
        jz      i5
        push    b       ;if tmp1 set, in a string
        mov     b,a     ; (so we have to ignore spaces)
        lda     tmp1
        ora     a
        mov     a,b
        pop     b
        jz      i3a
        cpi     '"'     ;we are in a string.
        jnz     i3      ;check for terminating quote
        xra     a       ;if found, reset "in string" flag
        sta     tmp1
        dcx     h
        mov     m,a     ;and stick a zero byte after the string
        inx     h       ;and go on to next arg
i3a:    cpi     ' '     ;now find the space between args
        jnz     i3
        dcx     h       ;found it. stick in a zero byte
        mvi     m,0
        inx     h
        jmp     i2      ;and go on to next arg
```

```
.5:     push    d       ;all done finding args. Set argv.
        lxi     b,argish-2      ;--- now push argv ---
        push    b

        mvi     b,nfcbs ;now initialize all the file info
        lxi     h,fdt   ;(just zero the fd table)
.6:     mvi     m,0
        inx     h
        dcr     b
        jnz     .6

        xra     a
        sta     ungetl  ;clear the push-back byte
        sta     lastc   ;and last character byte

        lhld    tmp2
        pchl            ;all done initializing.
;
; General purpose error value return routine:
;

error:  lxi     h,-1    ;general error handler...just
        ret             ;returns -1 in HL

;
; Here are file I/O handling routines, only needed under CP/M:
;

;
; Close any open files and reboot:
;

qzexit:
exit:
        mvi     a,7+nfcbs       ;start with largest possible fd
exit1:  push    psw             ;and scan all fd's for open files
        call    fgfd            ;is file whose fd is in A open?
        jc      exit2           ;if not, go on to next fd
        mov     l,a             ;else close the associated file
        mvi     h,0
        push    h
        call    close
        pop     h
exit2:  pop     psw
        dcr     a               ;and go on to next one
        cpi     7
        jnz     exit1

        jmp     exited          ;done closing; now reboot CP/M

;
; Close the file whose fd is 1st arg:
;

qzclose:
close:
        call    getlarg ;--- new ---
        call    setdma  ;library function just jumps here.
        call    matoh   ;get fd in A
        call    fgfd    ;see if it is open
        jc      error   ;if not, complain
        mov     a,m
        ani     4
        jz      close2  ;the file isn't open for write
        push    h       ;save fd table entry addr
        call    matoh   ;---was ma2toh --- move arg1 to A
        push    b
        call    fgfcb   ;get the appropriate fcb address
        xchg            ;put it in DE
        mvi     c,16    ;get BDOS function # for close
        call    bdos    ;and do it!
        pop     b
        pop     h
close2: mvi     m,0     ;close logically
        cpi     255     ;if 255 comes back, we got problems
        lxi     h,0
        rnz             ;return 0 if OK
        dcx     h       ;return -1 on error
        ret
;
; Determine status of file whose fd is in A...if the file
; is not open, return C flag set, else clear C flag:

fgfd:   call    setdma
        mov     d,a
        sui     8
        rc              ;if fd < 8, error
        cpi     nfcbs
        cmc             ;don't allow too big an fd either
        rc
        push    d
        mov     e,a     ;OK, we have a value in range. Now
        mvi     d,0     ;  see if the file is open or not
        lxi     h,fdt
        dad     d
        mov     a,m
        ani     1       ;bit 0 is high if file is open
        stc
        pop     d
        mov     a,d
        rz              ;return C set if not open
        cmc
        ret             ;else reset C and return

;
; Set up a CP/M file control block at HL with the file whose
; simple null-terminated name is pointed to by DE:
; Format for filename must be: "[write space][d:]filename.ext"
;

setfcb:
        call    setdma  ;set up an fcb at HL for filename at DE
        push    b

        call    lgwsp   ;ignore blanks and tabs
        mvi     b,8
        push    h
        inx     d
        ldax    d
        dcx     d
        cpi     ':'     ;default disk byte value is 0
        mvi     a,0     ; (for currently logged disk)
        jnz     setf1
        ldax    d       ;oh oh...we have a disk designator
        call    mapuc   ;make it upper case
        sui     '@'     ;and fudge it a bit
        inx     d
        inx     d
setf1:  mov     m,a
        inx     h
        call    patchnm ;now set filename and pad with blanks
        ldax    d
        cpi     '.'     ;and if an extension is given,
        jnz     setfcb2
        inx     d
setfcb2 mvi     b,3     ;set the extension and pad with blanks
        call    setnm
        xra     a       ;and zero the appropriate fields of the fcb
        mov     m,a
        lxi     d,20
        dad     d
        mov     m,a
        inx     h
;
        mov     m,a     ;zero the random record bytes
        inx     h
        mov     m,a
        inx     h
        mov     m,a
        pop     d
        pop     b
        ret

patchnm:
        call    setnm   ;another patch from "vsetfcb"
        jmp     setnm3

;
; This routine copes up to B characters from memory at DE to
; memory at HL and pads with blanks on the right:
;

setnm:  push    b
setnm1: ldax    d
        cpi     '*'     ;wild card?
        mvi     a,'?'   ;if so, pad with ? characters
        jz      pad2

setnm2: ldax    d
        call    legfc   ;next char legal filename char?
        jc      pad     ;if not, go pad for total of B characters
        mov     m,a     ;else store
        inx     h
        inx     d
        dcr     b
        jnz     setnm1  ;and go for more if B not yet zero
        pop     b
setnm3: ldax    d       ;skip rest of filename if B chars already found
        call    legfc
        rc
        inx     d
        jmp     setnm3

pad:    mvi     a,' '   ;pad with B blanks
pad2:   mov     m,a     ;pad with B instances of char in A
        inx     h
        dcr     b
        jnz     pad2
        pop     b
        ret

;
; Test if char in A is legal character to be in a filename:
;

legfc:  call    mapuc
        cpi     '.'     ; '.' is illegal in a filename or extension
        stc
        rz
        cpi     ';'     ;so is ':'
        stc
        rz
        cpi     7fh     ;delete is no good
        stc
        rz
        cpi     '!'     ;if less than exclamation pt, not legal char
        ret             ;else good enough

;
; Map character in A to upper case if it is lower case:
;

mapuc:  cpi     'a'
        rc
        cpi     'z'+1
        rnc
        sui     32      ;if lower case, map to upper
        ret

;
; Ignore blanks and tabs at text pointed to by DE:
;
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
igwsp:  dcx     d
igwsp1: inx     d
        ldax    d
        cpi     ' '
        jz      igwsp1
        cpi     9
        jz      igwsp1
        ret

;
; This routine does one of two things, depending
; on the value passed in A.
;
; If A is zero, then it finds a free file slot
;  (if possible), else returns C set.
;
; If A is non-zero, then it returns the address
; of the fcb corresponding to an open file whose
; fd happens to be the value in A, or C set if there
; is no file associated with fd.
;

fgfcb:  push    b
        call    setdma
        ora     a       ;look for free slot?
        mov     c,a
        jnz     fgfc2   ;if not, go away
        mvi     b,nfcbs ;yes. do it...
        lxi     d,fdt
        lxi     h,fcbt
        mvi     c,8
fgfc1:  ldax    d
        ani     1
        mov     a,c
        jnz     fgfc1a  ;found free slot?
        pop     b       ;yes. all done.
        ret
fgfc1a: push    d
        lxi     d,36    ;fcb length to accommodate random I/O
        dad     d
        pop     d
        inx     d
        inr     c
        dcr     b
        jnz     fgfc1
fgfc1b: stc
        pop     b
        ret             ;return C if no more free slots
fgfc2:  call    fgfd    ;compute fcb address for fd in A:
        jc      fgfc1b  ;return C if file isn't open

        sui     8
        mov     l,a     ;put (fd-8) in HL
        mvi     h,0
        dad     h       ;double it
        dad     h       ;4*a
        mov     d,h     ;save 4*a in DE
        mov     e,l
        dad     h       ;8*a
        dad     h       ;16*a
        dad     h       ;32*a
        dad     d       ;36*a
        xchg            ;put 36*a in DE
        lxi     h,fcbt  ;add to base of table
        dad     d       ;result in HL
        mov     a,c     ;and return original fd in A
        pop     b
        ret
setdma: push    d       ;just a preventative measure,
        push    b       ;since the default I/O buffer
        push    psw     ;tends to magically change
        push    h       ;around by itself when left
        mvi     c,26    ;in CP/M's hands !!
        lxi     d,tbuff
        call    bdos
        pop     h
        pop     psw
        pop     b
        pop     d
        ret
#endasm
```

```
#asm
; File I/O library for small-C -- Part I
;
; Source:  fiolib1
; Version: September 3, 1982
;
; Adapted from deff2.csm.  BDS C version 1.46,  3/22/82
; Original by Leor Zolman
; Modifications by Edward K. Ream
;
;
; Functions appearing in this file:
;
;       getchar kbhit   ungetch putchar putch   gets
;       setfcb  read    write   open    close   creat
;       unlink  seek    tell    rename
;       fabort  fcbaddr exit    bdos    bios
;

qzgetchar:
getchar:
        lda     ungetl  ;any character pushed back?
        ora     a
        mov     l,a
        jz      gch2
        xra     a       ;yes. return it and clear the pushback
        sta     ungetl  ;byte in C.CCC.
        mvi     h,0
        ret

gch2:   push    b
        mvi     c,conin
        call    bdos
        pop     b
        cpi     cntrlc  ;control-C ?
        jz      base    ;if so, reboot.
        cpi     1ah     ;control-Z ?
        lxi     h,-1    ;if so, return -1.
        rz
        mov     l,a
        cpi     cr      ;carriage return?
        jnz     gch3
        push    b
        mvi     c,conout        ;if so, also echo linefeed
        mvi     e,lf
        call    bdos
        pop     b
        mvi     l,newlin        ;and return newline (linefeed)..

gch3:   mvi     h,0
        ret


qzkbhit:
kbhit:
        lda     ungetl  ;any character ungotten?
        mvi     h,0
        mov     l,a
        ora     a
        rnz             ;if so, return true

        push    b
        mvi     c,cstat ;else interrogate console status
        call    bdos
        pop     b
        ora     a       ;0 returned by BDOS if no character ready
        lxi     h,0
        rz              ;return 0 in HL if no character ready
        inr     l       ;otherwise return 1 in HL
        ret


qzungetch:
ungetch:
        call    get1arg ;--- new ---
        lda     ungetl
        mov     l,a
        push    h
        call    ma1toh  ;--- was ma2toh ---
        sta     ungetl
        pop     h
        mvi     h,0
        ret

;this is a new function

qzputs:
puts:
        call    get1arg
        call    ma1toh
        push    b
        call    puts1
        pop     b
        ret

puts1:  mov     a,m
        cpi     0
        rz
        push    h               ;use putchar's interrupt logic
        mov     l,a
        push    h
        call    putchar
        pop     h
        pop     h
        inx     h
        jmp     puts1


qzputchar:
putchar:
        call    get1arg ;--- new ---
        call    ma1toh  ;get character in A
        push    b
        mvi     c,conout
        cpi     newlin  ;newline?
        jnz     put1    ;if not, just go put out the character
        mvi     e,cr    ;else...put out CR-LF
        call    bdos
        mvi     c,conout
        mvi     a,lf

put1:   mov     e,a
        call    bdos

put2:   mvi     c,cstat ;now, is input present at the console?
        call    bdos
        ora     a
        jnz     put3
        pop     b       ;no...all done.
        ret
put3:   mvi     c,conin ;yes. sample it (this will always echo the
        call    bdos    ;       character to the screen, alas)
        cpi     cntrlc  ;is it control-C?
        jz      base    ;if so, abort and reboot
        pop     b       ;else ignore it.
        ret


qzputch:
putch:
        call    get1arg ;--- new ---
        call    ma1toh
```

```
        push    b
        mvi     c,conout
        mov     e,a
        cpi     newlin
        jnz     putch1  ;if not newline, just put it out
        mvi     e,cr    ;else put out CR-LF
        call    bdos
        mvi     c,conout
        mvi     e,lf
putch1: call    bdos
        pop     b
        ret


qzgets:
gets:
        call    getlarg ;--- new ---
        call    maltoh  ;get destination address
        push    b       ;save BC
        push    h
        push    h
        lxi     h,-150  ;use space below stack for reading line
        dad     sp
        push    h       ;save buffer address
        mvi     m,85h   ;Allow a max of about 135 characters
        mvi     c,getlin
        xchg            ;put buffer addr in DE
        call    bdos    ;get the input line
        mvi     c,conout
        mvi     e,lf    ;put out a LF
        call    bdos
        pop     h       ;get back buffer address
        inx     h       ;point to returned char count
        mov     b,m     ;set B equal to char count
        inx     h       ;HL points to first char of line
        pop     d       ;DE points to start destination area
copy1:  mov     a,b     ;copy line to start of buffer
        ora     a
        jz      gets2
        mov     a,m
        stax    d
        inx     h
        inx     d
        dcr     b
        jmp     copy1

gets2:  xra     a       ;store terminating null
        stax    d
        pop     h       ;return buffer address in HL
        pop     b
        ret


qzsetfcb:
        call    get2args        ;--- was arghak ---
        push    b
        lhld    arg2    ;get pointer to name text
igsp:   mov     a,m
        inx     h
        cpi     ' '
        jz      igsp
        cpi     tab
        jz      igsp
        dcx     h
        xchg            ;set DE pointing to 1st non-space char
        lhld    arg1    ;get --> fcb area
        call    setfcb  ; do it
        lxi     h,0     ;all OK.
        pop     b
        ret


qzread:
read:
        call    get3args        ;--- was arghak ---
        lda     arg1
        call    fgfcb
        jc      error   ;error if illegal fd
        mov     a,m
        ani     2       ;open for read?
        jz      error   ;error if not
        push    b
        lda     arg1
        call    fgfcb
        shld    tmp2    ;tmp2 will hold dma addr
        lxi     h,0     ;count of # of successful sectors read
        shld    tmp2a   ; will be kept at tmp2a
read2:  lhld    arg3    ;done?
        mov     a,h
        ora     l
        jz      read4

read2a: lhld    arg2    ;else read another sector
        xchg            ;DE is dma addr
        mvi     c,sdma
        call    bdos    ;set DMA
        lhld    tmp2
        xchg            ;DE is fcb addr
        mvi     c,reads
        push    d       ;save de so we can fudge nr field if
        call    bdos    ;we stop reading on extent boundary...
        pop     d               ; CP/M sucks!
        cpi     2
        pop     b
        jz      error   ;if error, abort
        push    b
        cpi     1
        jnz     read6   ;EOF?

read3:  lxi     h,32    ;yes. are we on extent boundary?
        dad     d       ;if so, adjust for CP/M's stupidity here
        mov     a,m     ;by turning an 80h sector count into 00h.
        cpi     80h
        jnz     read4
        mvi     m,0     ;yes, reset nr to 0...
read4:  lhld    tmp2a
read5:  pop     b
        ret

read6:  lhld    arg3
        dcx     h
        shld    arg3
        lhld    arg2
        lxi     d,128
        dad     d
        shld    arg2
        lhld    tmp2a
        inx     h
        shld    tmp2a
        jmp     read2


qzwrite:
write:
        call    get3args        ;--- was arghak ---
        lda     arg1
        call    fgfcb
        jc      error
        mov     a,m
        ani     4
        jz      error
        push    b
        lda     arg1
        call    fgfcb
        shld    tmp2
        lxi     h,0
        shld    tmp2a
        lxi     d,tbuff         ;80 for normal CP/M, else 4280
        mvi     c,sdma
        call    bdos

writ1:  lhld    arg3    ;done yet?
        mov     a,h
        ora     l
        lhld    tmp2a   ;if so, return count
        jz      writ3
        lhld    arg2    ;else copy next 128 bytes down to tbuff
        lxi     d,tbuff ;80 for normal CP/M, else 4280
        mvi     b,128
writ2:  mov     a,m
        stax    d
        inx     h
        inx     d
        dcr     b
        jnz     writ2
        shld    arg2    ;save -> to next 128 bytes
        lhld    tmp2    ;get addr of fcb
        xchg
        mvi     c,write ;go write
        call    bdos
        ora     a       ;error?
        lhld    tmp2a   ;if so, return # of successfully written
        jnz     writ3   ;  sectors.

        inx     h       ; else bump successful sector count,
        shld    tmp2a
        lhld    arg3    ; debump countdown,
        dcx     h
        shld    arg3
        jmp     writ1   ; and go try next sector
writ3:  pop     b
        ret


qzopen:
open:
        call    get2args        ;--- was arghak ---
        xra     a
        call    fgfcb   ;any fcb's free?
        jc      error   ;if not, error
        sta     tmp
        xchg
        lhld    arg1
        xchg
        push    b
        call    setfcb
        mvi     c,openc
        call    bdos
        cpi     errorv  ;successful open?
        pop     b
        jz      error   ;if not, error
        lda     tmp
        call    fgfd    ;get HL pointing to fd table entry
        lda     arg2
        ora     a       ;open for read?
        mvi     d,3
        jz      open1
        dcr     a
        mvi     d,5
        jz      open1   ;write?
        dcr     a
        jnz     error   ;else must be both or bad mode.
        mvi     d,7
open1:  mov     m,d
        lda     tmp
        mov     l,a
        mvi     h,0
        ret


qzcreat:
creat:
        call    getlarg ;--- was arghak ---
        lhld    arg1
        push    b
```

(Continued on next page)

RED - Listing

(Listing continued, text begins on page 62)

```
        push    h
        call    unlink  ;erase any old versions of file
        pop     d
        mvi     c,create
        lxi     d,fcb
        call    bdos
        cpi     errorv
        pop     b
        jz      error

        lhld    arg1    ;--- push these in reverse order ---
        push    h
        lxi     h,2     ;open for read/write
        push    h

        call    open
        pop     d
        pop     d
        ret
qzunlink:
unlink:
        call    getlarg ;--- new ---
        call    malloh
        push    b
        xchg
        lxi     h,fcb
        call    setfcb
        mvi     c,delc
        call    bdos
        lxi     h,0
        pop     b
        ret
qzseek:
seek:
        call    get3args        ;--- was srghak ---
        lda     arg1
        call    fgfcb
        jc      error   ;error if file not open
        push    b
        push    h       ;save fcb address

        lhld    arg1
        push    h
        call    tell    ;get r/w pointer position for the file
        pop     d

        xchg            ;put present pos in DE
        lda     arg3
        lhld    arg2    ;get offset in HL
        ora     a       ;absolute offset?
        jz      seek2   ;if so, offset is new position
        dad     d       ;else add offset to current position
seek2:  mov     a,l     ;convert to extent and sector values
        rlc
        mov     a,h
        ral
        ani     7fh
        sta     tmp
        xthl
        lxi     d,12
        push    h
        dad     d
        cmp     m       ;jumping over extent boundary?
        jz      seek5
        xthl            ;yes.
        xchg
        mvi     c,closec        ;close old extent
        push    d
        call    bdos
        pop     d
        pop     h
        cpi     errorv
        jnz     seek4
seek3:  pop     d
        pop     b
        jmp     error

seek4:  lda     tmp
        mov     m,a
        push    d
        mvi     c,openc ;and open new one.
        call    bdos
seek5:  pop     d
        cpi     errorv
        jz      seek3
        lxi     h,32    ;and set nr field
        dad     d
        pop     d
        mov     a,e
        ani     7fh
        mov     m,a
        xchg            ;return new sector # in HL
        pop     b
        ret

qztell:
tell:
        call    getlarg ;--- new ---
        call    malloh  ;get fd value in A
        call    fgfcb
        jc      error
        push    b
        lxi     d,12
        dad     d
        mov     b,m     ;put extent # in B
        lxi     d,20
        dad     d
        mov     c,m     ;put sector # in C
        xra     a       ;rotate extent right one bit
                        ;old b0 --> Carry
        mov     a,b
        rar
        mov     h,a     ;rotated value becomes high byte of
                        ;tell position
        mvi     a,0     ;rotate b0 of extent into A
        rar
        mov     b,a     ;save rotated extent number in B
        add     c       ;add rotated extent number to sector #
        mov     l,a     ;and result becomes low byte of
                        ;tell position
        mov     a,c     ;if both rotated extent # and sector #
                        ;has bit 7 hi,
        add     b       ;then the sum had an overflow, so...
        jp      tell2
        inr     h       ;bump position number by 256
tell2:  pop     b       ;and all done.
        ret


qzrename:
rename:
        call    get2args        ;--- was srghak ---
        push    b
renam:  lhld    arg1
        xchg
        lxi     h,wfcb
        call    setfcb
        lhld    arg2
        xchg
        lxi     h,wfcb+16
        call    setfcb
        lxi     d,wfcb
        mvi     c,renc
        call    bdos
        pop     b
        cpi     errorv
        jz      error
        lxi     h,0
        ret
wfcb:   ds 33


qzfabort:
fabort:
        call    getlarg ;--- new ---
        call    malloh
        call    fgfd
        jc      error
        mvi     m,0     ;clear entry in fd table
        lxi     h,0
        ret


qzfcbaddr:
fcbaddr:
        call    getlarg ;--- new ---
        call    malloh
        call    fgfd    ;is it an open file?
        jc      error
        call    malloh
        call    fgfcb   ;get fcb addr in HL
        ret


bdos    equ     5
qzbdos:
        call    get2args        ;--- was srghak ---
        push    b
        lda     arg1    ;get C value
        mov     c,a
        lhld    arg2    ;get DE value
        xchg            ;put in DE
        call    bdos    ;make the bdos call
        pop     b
        ret             ;and return to caller


qzbios:
bios:
        call    get2args        ;--- was srghak ---
        push    b
        lhld    base+1  ;get addr of jump table + 3
        dcx     h       ;set to addr of first jump
        dcx     h
        dcx     h
        lda     arg1    ;get function number (1-35)
        mov     b,a     ;multiply by 3
        add     a
```

```
        add     b
        mov     e,a     ;put in DE
        mvi     d,0
        dad     d       ;add to base of jump table
        push    h       ;and save for later
        lhld    arg2    ;get value to be put in BC
        mov     b,h     ;and put it there
        mov     c,l
        lxi     h,retadd        ;where call to bios will return to
        xthl            ;get address of vector in HL
        pchl            ;and go to it...
retadd: mov     l,a     ;all done. now put return value in HL
        mvi     h,0
        pop     b
        ret             ;and return to caller
#endasm
```

```
#asm
; Extra small-C run time library
;
; Source:  xlib
; Version: August 29, 1982
;
; Adapted from ceff2a.asm.  BDS C version 1.46,  3/22/80
; Original by Leor Zolman
; Modified by Edward K. Ream
;

qzsetjmp:
        call    getlarg ;--- new ---
        call    matoh

        mov     m,c     ;save BC (not really needed in small-C)
        inx     h
        mov     m,b
        inx     h

        xchg            ;save SP
        lxi     h,0
        dad     sp
        xchg
        mov     m,e
        inx     h
        mov     m,d
        inx     h

        pop     d       ;save return address
        push    d
        mov     m,e
        inx     h
        mov     m,d

        lxi     h,0     ;and return 0
        ret

;longjmp(buffer, return_value)
;
;       unlike the BDS C version of this routine,
;       the return_value is REQUIRED
;

qzlongjmp:
        call    get2args        ;--- new ---
        call    matoh   ;get buffer address

        mov     c,m     ;restore BC
        inx     h
        mov     b,m
        inx     h

        mov     e,m     ;restore SP...first put it in DE
        inx     h
        mov     d,m
        inx     h

        shld    temp    ;save pointer to return address
        call    ma2toh  ;get return value

        xchg            ;put return val in DE, old SP in HL
        sphl            ;restore SP with old value

        pop     h       ;pop return address off stack
        lhld    temp    ;get back ptr to return address

        mov     a,m
        inx     h
        mov     h,m
        mov     l,a     ;HL holds new return address

        xchg            ;ret addr in DE, return value in HL
        push    d       ;push return address on stack
        ret             ;and return...

temp:   ds 2
#endasm
```

End Listing Two