Spring 5-1999

HTML Editor for UNIX Terminals in C

Bradley Alan Edmonson
University of Tennessee - Knoxville

UNIVERSITY HONORS PROGRAM

SENIOR PROJECT - APPROVAL

Name: Bradley A Edmonson

College: Arts & Sciences Department: Computer Science

Faculty Mentor: Dr. James Plank

PROJECT TITLE: HTML Editor for UNIX Terminals in C

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: _____, Faculty Mentor

Date: 5/12/99

Comments (Optional):

HTML Editor for UNIX Terminals in C

Bradley A. Edmonson
12 May 1999

ABSTRACT

With the ushering in of the information age came the unprecedented ability for a person to publish any work at negligible costs on the World Wide Web. Publishing on the web, however, revolves around *hypertext markup language*, or HTML. While relatively simple when contrasted with other programming languages, writing in HTML can be a daunting task for a true amateur to undertake. Thus, a number of commercial products have hit the market to mask the details of HTML and streamline the process of web publishing. These products, however, frequently do not have the interest of the casual user in mind, requiring a fairly substantial investment to use. A free, easy to use editor is ideal for this small yet still important group of people.

Realizing UNIX-based machines power many of the Web's servers, a UNIX HTML editor is a candidate to fill this void. Since many users of UNIX systems have access to the machine only through terminal programs, indeed these users are representative of the casual users mentioned above, a text-based HTML editor designed for use with UNIX terminals could potentially be a very useful product. This project attempts to create both a useful and educational web development tool for the casual user. The utility is designed to be very simple and guide the user through the more unusual bits of HTML syntax while at the same time preserving the freedom and power of the language. None of the syntax is hid from the user, either; shortcuts simply allow it to be inserted much more quickly and with little to no possibility for error. By having the syntax continuously presented to the user, it is the author's hope that knowledge of the language will be encouraged instead of veiled.

INTRODUCTION

Not everyone can afford to participate in the so-called "Information Revolution." While businesses and many private individuals have embraced the sweeping changes brought upon by information technology, many would just assume to live their lives the same way they had been without any regard to the "Revolution" at all. Caught in between are those who are familiar with the technology but cannot really exploit it in any useful way due to lack of technical expertise or the time to gather such knowledge. These casual users may not be numerous, but their needs should still be addressed.

At the core of the growing field of information science is the Internet. Though known by any number of names, the Internet is ultimately just files being shared by computers around the world. In order for this file sharing to take place, some sort of standard format is necessary so that different types of machines will be able to communicate properly. With the advent of the web browser, the standard format became hypertext markup language or HTML. While a very simple "language" to learn, writing in HTML can be a daunting task for someone unfamiliar with computer programming and the thought processes involved. Thus, a simple, easy to use HTML editor would be a powerful tool for these casual users.

It is a well-known fact that a large number of web servers are powered by UNIX or UNIX-clone operating systems. Since many casual users would be making use of the server of a company, school, or other host, it makes sense that an HTML editor designed

for them would run on UNIX. Often times, the best access these users might be able to get to the machine itself is from a terminal program which are usually text-based. Therefore, the editor would also need to take this into account, making a text-based editor ideal.

The intent of this project was to create such an editor that would be available for free download and usage. The specific group of users the utility was created for were students at both the high school and collegiate levels. A complete copy of the written code has been attached to this report in the form of an appendix.

BACKGROUND DATA

The C language was chosen for this project due to its relative ubiquity in the programming community as well as the availability of the curses library. The curses library grants a programmer a fairly optimized method of manipulating a terminal screen. Curses allows, amongst other things, the creation and manipulation of windows on a terminal, key elements to completing this project.

Two libraries written by Dr. James Plank, University of Tennessee-Knoxville, were also made use of. The first library, the *dlist* library, abstracts an optimized doubly-linked list data storage utility for the programmer. By allowing dynamic allocation of memory, this data structure was ideal for the creation and storage of lines of HTML code since random

access is not necessary in the scope of a text editor. At most, one would move from one line to the next or previous line through use of the arrow keys on the keypad, and this motion is most easily accomplished using a doubly-linked list. The second library, the *fields* library, greatly simplifies the processing of input in C. Although used sparingly in this project, the library was necessary for processing the lines of files opened by the editor instead of attempting to do so on a character-by-character basis.

The original code written for the project is contained in five files: *main.h*, *main.c*, *process.c*, *menus.c*, and *other.c*. The header file, *main.h*, contains included libraries, global variables, constant definitions, and function prototypes for the functions contained in the other four files. *Main.c* initializes the basic interface and processes the program's main menu. *Process.c* deals with the actual processing of input from the user and the implementation of the various options associated with HTML. *Menus.c* contains the functions that control the display of the three accessory windows. *Other.c* contains a number of functions that assist in input/output processes as well as functions dealing with updating the data structure and the main window.

IMPLEMENTATION

The basic interface consists of four windows: *topwin*, *botwin*, *meswin*, and *cenwin*. *Topwin* consists of two lines at the top of the screen used for displaying the name of the file currently being worked on. *Botwin* encompasses three lines at the bottom of the

screen and displays the options available to the user during the actual process of editing. *Meswin* is a one-line window directly above *botwin* used for giving simple messages to the user as well as for retrieving input from the user not intended to go directly into the document. *Cenwin* is the size of all the lines in between *topwin* and *meswin*, depending on the size of the terminal. *Cenwin* takes up the majority of the screen and is where the document can be found during the process of editing. All of these windows are declared as global variables due to the fact that one or more of them are used in some form by nearly all of the functions in the program.

The actual procedure that the program uses is very simple. Once the editor has been started, the function *pro_line* simply waits until the user strikes a key. Once it has this character, it compares it with a number of possible options. First, it looks at menu options: the special keys **<CTRL>-E** (Exit), **<CTRL>-A** (Save), and **<CTRL>-T** (HTML Options Menu). Next, *pro_line* checks it against the possible keypad strokes: left, right, up, and down. Control symbols such as *newline* and *backspace* are next, and if it finds no match for any of these it assumes that the character was just a regular piece of text meant to be output to the screen.

The HTML Options menu is fairly straightforward as well. It waits for a key to be pressed just as its predecessor did, then executes the option associated with said key. Any HTML option can be encoded, though at the point of this writing only the most major and standard uses were actually implemented into the menu. These include links, images, text formatting, and several others.

The interesting part of the editing procedure occurs when trying to maintain the data structure that holds all the lines that the user has entered into the document while at the same time keeping this consistent with what has been displayed on the screen. The editor solves this problem by using a doubly-linked list to store all the lines in the opened document and keeping track of the exact cursor location in the data structure with a pair of pointers. The pointer *ln* keeps track of the line number while *len* is used to locate the proper position within the line. Anytime the cursor is moved, a character is printed, or any other action takes place on the screen, these pointers are adjusted as needed to ensure that the location within the data structure remains consistent within its on-screen image.

For more details about the behavior of individual functions, please refer to the comments in the actual code located in the appendix.

STYLE

Since this utility was created for a casual user with little or no experience with programming or computer languages, simplicity of design and education about HTML were principle during the design procedure. The resulting user interface is simple to use without leeching the power of HTML. Furthermore, none of the syntax is hidden from the user's view, shortcuts only make enacting it simpler. This allows the user to learn the language even as he or she is exposed to it for the first time.

Now we will create a very simple web page using the editor in order to illustrate its mechanics. When the program is first run from the command line, the user sees the following screen.

(Note: All screenshots have had blank lines removed in the interest of saving space.)

```
UNIX Term HTML Editor - Brad Edmonson
--------------------------------------------------------------------------------

        N - New HTML Document

        O - Open HTML Document

        X - Exit Editor

        H - Help
```

Figure 1: Initial screen.

At this point, the user has several options, but since we wish to create a new page from scratch, we press the 'n' key for 'New HTML Document'.

```
DOCUMENT:  NEW FILE
--------------------------------------------------------------------------------
<HTML>
<BODY>


^E: Exit(No Save)        ^A: Save        ^T: HTML Edit Options
```

Figure 2: Initial edit screen.

Now we are at the main text edit screen. Note that the initial HTML tags '<HTML>' and '<BODY>' are automatically entered into the new document. The cursor is located

directly underneath the '<BODY>' tag and we are ready to begin drawing up our web page.

```
DOCUMENT:  NEW FILE
------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX HTML Editor.<br>
<p>


^E: Exit(No Save)        ^A: Save          ^T: HTML Edit Options
```

Figure 3: Edit screen after a few words are typed.

To arrive with this output, we simply typed out "This is a sample page created using the UNIX HTML Editor" then pressed enter twice. Notice that the HTML tags '
' and '<p>' for 'break' and 'paragraph' respectively are automatically entered upon pressing enter. Now we will insert a link to the author's homepage.

```
DOCUMENT:  NEW FILE
------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is


Enter a Selection:
E: End HTML Edit         L: Hyperlink     F: Font Options I: Italics
B: Bold  P: Picture       C: Center
```

Figure 4: HTML options menu.

Bringing up the HTML options menu, we now press 'L' in order to insert a hyperlink.

```
DOCUMENT:  NEW FILE
------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="



Address of the link?  http://www.cs.utk.edu/~edmonson
E: End HTML Edit          L: Hyperlink     F: Font Options I: Italics
B: Bold P: Picture        C: Center
```

Figure 5: The user is prompted for the address of the link to be inserted.

The program now automatically inserts the first part of the HTML tag, '<A HREF="". We are then prompted to enter the address of the web page in the message window. The address is typed in and we strike enter.

```
DOCUMENT:  NEW FILE
------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="http://www.cs.utk.edu/~edmonson">Brad Edmonson



Insert link text now, choose 'Hyperlink' again to close
^E: Exit(No Save)         ^A: Save          ^T: HTML Edit Options
```

Figure 6: The inserted hypertext link.

The rest of the address is automatically written to the document, and we are told to go ahead and enter the text of the link, choosing the 'hyperlink' option once again when we are done. We could use an image or some sort or formatted text as a link, but we will just enter the author's name, "Brad Edmonson", then close the link text.

```
DOCUMENT:  NEW FILE
------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="http://www.cs.utk.edu/~edmonson">Brad
Edmonson</A><br>
<p>


^E: Exit(No Save)        ^A: Save          ^T: HTML Edit Options
```

Figure 7: The completed hypertext link.

With the completion of the link, the HTML tag '</A>' is used to close it off in the file.

Now we would like to save our file, so we press 'Control-A' in order to save.

```
DOCUMENT:  NEW FILE

------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="http://www.cs.utk.edu/~edmonson">Brad
Edmonson</A><br>
<p>



FILENAME TO SAVE AS: sample.html
^E: Exit(No Save)        ^A: Save          ^T: HTML Edit Options
```

Figure 8: The user is prompted for the filename to save the document as.

Once again, we are prompted to enter the filename. We decide to save the file as sample.html.

```
DOCUMENT:  sample.html
-------------------------------------------------------------------------
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="http://www.cs.utk.edu/~edmonson">Brad
Edmonson</A><br>
<p>


File Saved
^E: Exit(No Save)        ^A: Save        ^T: HTML Edit Options
```

Figure 9: Newly saved file.

Now we are told that the file has successfully been saved, and the name of the file in the upper window has been changed from 'NEW FILE' to 'sample.html'. However, this is simply the display on the screen in the editor. How does the file created by outputting the data structure compare with what we saw on the screen? Exiting the program, we bring up the text of the file sample.html

```
<HTML>
<BODY>
This is a sample page created using the UNIX Terminal HTML Editor.<br>
<p>
The author is <A HREF="http://www.cs.utk.edu/~edmonson">Brad
Edmonson</A><br>
<p>

</BODY>
</HTML>
```

Figure 10: File sample.html.

As we can see, the file is identical to the document we saw on the screen except for the fact that the two initial tags, '<HTML>' and '<BODY>' have been closed off with their counterparts at the end of the file. The file is now finished and ready to be viewed by any

web browser from text-based versions such as Lynx to more complex products such as Netscape or Internet Explorer. Obviously, one could use this editor to create pages much more complicated than this simple example.

ANALYSIS

The editor accomplishes what was originally intended in a reasonably optimized manner. The primary goal of simplicity of use was achieved through the use of shortcut keys, constantly updated menus, and the continuous suggestions of the message window. The secondary goal of educating the user was achieved through the display of all HTML tags even though the user is not directly responsible for inputting every letter of these tags. The interface encourages the user to memorize or modify the tags inserted by the editor itself. An additional advantage to the editor is the easy manner by which it can be expanded to include new options. Any new HTML options desired can be added to the program in a matter of minutes as a result of good structural design. In terms of computing power, the editor lies well within acceptable limits. Although written on a powerful UNIX machine, the editor uses at its peak only around 1.5% of the processor, so a much less powerful machine would still handle it easily. The dynamic allocation of the data structure makes the matter of memory trivial, limited only by the size of the memory of the machine on which the program is run. The combination of these factors qualify the project as a reasonable solution to the stated problem.

Appendix

A:edmonson Job: main.h Date: 1999-05-11-15:14:21

```
/* Include statements */
#include <stdio.h>
#include <string.h>
#include <curses.h>
/* Dlist and fields libraries created by Dr. James Plank, */
/*  University of Tennessee at Knoxville, and are used    */
/*  with his permission.                                  */
#include "dlist.h"
#include "fields.h"

/* Constants */
#define MAX_FSIZE 200
#define MAX_LINES 100000
#define BUFFER 5

/* Global variables */
/* Windows */
WINDOW *screen;
WINDOW *topwin;
WINDOW *cenwin;
WINDOW *botwin;
WINDOW *meswin;
/* Other Globals - HTML Tags and the Data Structure */
int italic_tag,bold_tag,link_tag,font_tag,center_tag;
Dlist lineset,cline;

/* process.c function prototypes */
int pro_new(char*);
int pro_line(char*);
int pro_html_line(char*,int,int*,int*,int*);

/* menus.c function prototypes */
void menu_top_init();
void menu_top_editfile(char*);
void menu_cen_init();
void menu_bot_init();
void menu_bot_editfile();
void menu_bot_edithtml();
void put_cursor_here(WINDOW*);

/* other.c function prototypes */
void goodbye_cruel_world();
char *get_name(char*,char*,int);
void message(char*);
void copy_to_file(Dlist,int,FILE*);
char *line_right(char*,int);
void to_the_right(char*,int,int);
int blackout(char*,int);
int squeezein(char*,char*,int*,int*,int*);
void spillcheck(int,char*,int*,int*,int*);
```

A:edmonson Job: main.c Date: 1999-05-11-15:14:08

```
/* Untitled at this point in time                          */
/*                                                         */
/* Brad Edmonson - HTML Editor for UNIX Terminals          */

/* This portion of the code sets up the basic interface */
/*  using curses to make several windows for menus and  */
/*  text entering.                                      */

#include "main.h"

main()
{
        int in;
        char *name;
        FILE *fileout;

/* Curses initialization procedures */
        screen=initscr();
        topwin=newwin(2,COLS,0,0);
        cenwin=newwin(LINES-6,COLS,2,0);
        meswin=newwin(1,COLS,LINES-4,0);
        botwin=newwin(3,COLS,LINES-3,0);
        scrollok(topwin,1);
        scrollok(cenwin,1);
        scrollok(meswin,1);
        scrollok(botwin,1);
        clearok(topwin,1);
        clearok(cenwin,1);
        clearok(meswin,1);
        clearok(botwin,1);
        keypad(cenwin,1);
        keypad(meswin,1);
        noecho();

        menu_top_init();
        menu_bot_init();
        menu_cen_init();

        while(1){
                in=wgetch(cenwin);
                if(in!=ERR){

/* New File Option */
                        if(in=='N' || in=='n'){
                                menu_top_editfile("NEW FILE");
                                menu_bot_editfile();
                                message("");
                                pro_new(NULL);
                                message("");
                                menu_top_init();
                                menu_bot_init();
                               }

/* Open File Option */
                        else if(in=='O' || in=='o'){
                                name=strdup(get_name("FILE TO OPEN: ","*.html",1));
                                fileout=fopen(name,"r");
                                if(fileout==NULL)
                                        message("File Not Found");
                                else{
                                        fclose(fileout);
                                        menu_top_editfile(name);
                                        menu_bot_editfile();
                                        message("");
                                        pro_new(name);
                                        message("");
```

```
                                        free(name);
                                        menu_top_init();
                                        menu_bot_init();
                                       }
                                }

/* Exit Program Option */
                        else if(in=='X' || in=='x'){
                                goodbye_cruel_world();
                                }
                        menu_cen_init();
                       }
                }
}
```

A:edmonson Job: process.c Date: 1999-05-11-15:14:31

```
#include "main.h"

/* pro_new: serves as a go-between between the main interface */
/*  and the actual text-edit interface                        */
int pro_new(char *filename)
{
        int buffsize;
        FILE *fileout;

        wclear(cenwin);
        wrefresh(cenwin);

        while(1){
                buffsize=pro_line(filename);
                if(buffsize==-1){
                        return(1);
                       }
                wrefresh(cenwin);
               }
}

/* pro_line: process text as well as the options on the main */
/*  edit menu.  Deals with any possible input such as keypad */
/*  strokes, regular characters, and command characters.     */
int pro_line(char *filename)
{
        int in,last_br,len,ln,maxln,tag,scrln;
        char *name,*line;
        char input[2];
        IS is;
        FILE *file;

/* Initialize the line buffer */
        line=(char*)malloc((sizeof(char)*COLS)+100);

/* New File Procedure */
        if(filename==NULL){
                tag=0;
                lineset=make_dl();
                dl_insert_b(lineset,"<HTML>\n");
                dl_insert_b(lineset,"<BODY>\n");
                dl_insert_b(lineset,"");
                wprintw(cenwin,"<HTML>\n<BODY>\n");
                cline=lineset->flink->flink->flink;
                maxln=2;
                len=0;
                ln=2;
                scrln=2;
                wrefresh(cenwin);
                strcpy(line,cline->val);
               }

/* Opened File Procedure */
        else{
                tag=1;
                lineset=make_dl();
                file=fopen(filename,"r");
                is=new_inputstruct(filename);
                maxln=-1;
                while(get_line(is)>=0){
                        maxln++;
                        dl_insert_b(lineset,strdup(is->text1));
                        if(maxln<(LINES-7))
                                wprintw(cenwin,"%s",is->text1);
                       }
                jettison_inputstruct(is);
```

```
                cline=lineset->flink;
                ln=0;
                scrln=0;
                len=0;
                wmove(cenwin,0,0);
                wrefresh(cenwin);
                strcpy(line,cline->val);
               }

/* Initialize HTML Tags to "No" */
        italic_tag=0;
        bold_tag=0;
        link_tag=0;
        font_tag=0;
        center_tag=0;
        last_br=0;

/* Start the infinite character-processing loop, ignore */
/*  if you encounter and error and get another one.     */
        while(1){
        in=wgetch(cenwin);
        if(in!=ERR){

/* Implement the menu options first */

/* EXIT */
                if(in==5){

/* Copy the current line to the data structure */
                        cline->val=strdup(line);

/* Determine if they would like to save, get the filename if yes */
                        message("Would you like to save before exiting? (Y or N)");
                        in=wgetch(meswin);
                        if(in=='Y' || in=='y'){
                                if(filename!=NULL)
                                        name=strdup(get_name("FILENAME TO SAVE AS: ",
                                else{
                                        name=strdup(get_name("FILENAME TO SAVE AS: ",
                                        filename=strdup(name);
                                       }
                                file=fopen(name,"wb");
                                free(name);
                               }

/* Automatic Backup in case they meant to hit yes */
                        else{
                                file=fopen(".temp.html","wb");
                               }

/* Copy over to the appropriate file */
                        copy_to_file(lineset,maxln,file);

/* Close off the HTML file, but only if we were dealing with a new file */
                        if(tag==0)
                                fwrite("\n</BODY>\n</HTML>",sizeof(char),16,file);
                        fclose(file);
                        return(-1);
                       }

/* SAVE - works pretty much the same as above */
                else if(in==1){
                        cline->val=strdup(line);
                        if(filename!=NULL)
                                name=strdup(get_name("FILENAME TO SAVE AS: ",filenam
                        else{
```

```
                                name=strdup(get_name("FILENAME TO SAVE AS: ","*.html"
                                filename=strdup(name);
                                menu_top_editfile(filename);
                               }
                        file=fopen(name,"wb");
                        copy_to_file(lineset,maxln,file);
                        if(tag==0)
                                fwrite("\n</BODY>\n</HTML>",sizeof(char),16,file);
                        fclose(file);
                        free(name);
                        message("File Saved");
                       }

/* BACKSPACE */
                else if(in=='\b'){
                        len=blackout(line,len);
                       }

/* HTML Options Command - just call the right function, modify */
/*  the bottom menu                                            */
                else if(in==20){
                        menu_bot_edithtml(botwin);
                        put_cursor_here(cenwin);
                        last_br=pro_html_line(line,last_br,&maxln,&len,&ln);
                        menu_bot_editfile(botwin);
                        put_cursor_here(cenwin);
                       }

/* KEYPAD MANIPULATION */

/* Left, go left */
                else if(in==KEY_LEFT){
                        wmoveprevch(cenwin);
                        len--;
                       }

/* Right, go right unless you're at the end of the line */
                else if(in==KEY_RIGHT){
                        if(len<strlen(line)){
                                wmovenextch(cenwin);
                                len++;
                               }
                       }

/* Down, go down unless at the bottom of the file.  scrln keeps */
/*  track of the current line number in the window, not overall */
                else if(in==KEY_DOWN){
                        if(ln!=maxln){
                                cline->val=strdup(line);
                                cline=cline->flink;
                                strcpy(line,cline->val);
                                ln++;
                                if(scrln>=(LINES-8)){
                                        wscrl(cenwin,1);
                                        wrefresh(cenwin);
                                        wmove(cenwin,scrln,0);
                                        wrefresh(cenwin);
                                        wprintw(cenwin,"%s",cline->val);
                                        wrefresh(cenwin);
                                       }
                                else
                                        scrln++;
                                if(len>strlen(line))
                                        len=strlen(line);
                                wmove(cenwin,scrln,len);
                                wrefresh(cenwin);
```

```
                                }
                        }

/* Up, go up unless at the top of the file.  scrln here again */
                else if(in==KEY_UP){
                        if(ln!=0){
                                cline->val=strdup(line);
                                cline=cline->blink;
                                strcpy(line,cline->val);
                                ln--;
                                if(scrln==0){
                                        wscrl(cenwin,-1);
                                        wrefresh(cenwin);
                                        wmove(cenwin,scrln,0);
                                        wrefresh(cenwin);
                                        wprintw(cenwin,"%s",cline->val);
                                        wrefresh(cenwin);
                                       }
                                else
                                        scrln--;
                                if(len>strlen(line))
                                        len=strlen(line);
                                wmove(cenwin,scrln,len);
                                wrefresh(cenwin);
                               }
                        }

/* TEXT PROCESSING */

/* New line character - last_br keeps track of how many have been */
/*  struck in a row, two in a row means paragraph break, else     */
/*  just insert a regular break.                                  */
                else if(in=='\n'){

/* Paragraph break, note that it copies over the line before going on */
                        if(last_br>0){
                                len=squeezein(line,"<p>",&len,&ln,&maxln);
                                wprintw(cenwin,"\n");
                                wrefresh(cenwin);
                                strncat(line,"\n",1);
                                free(cline->val);
                                cline->val=strdup(line);
                                dl_insert_b(lineset,"");
                                cline=cline->flink;
                                len=0;
                                ln++;
                                maxln++;
                                if(scrln>=(LINES-8));
                                else scrln++;
                                strcpy(line,cline->val);
                               }

/* Regular break, copies here also */
                        else{
                                len=squeezein(line,"<br>",&len,&ln,&maxln);
                                wprintw(cenwin,"\n");
                                wrefresh(cenwin);
                                strncat(line,"\n",1);
                                free(cline->val);
                                cline->val=strdup(line);
                                dl_insert_b(lineset,"");
                                cline=cline->flink;
                                len=0;
                                last_br++;
                                ln++;
                                maxln++;
```

```
                                if(scrln>=(LINES-6));
                                else scrln++;
                                strcpy(line,cline->val);
                               }
                        wrefresh(cenwin);
                       }

/* If it wasn't any of that, just pass it to squeezein by itself */
/*  since it must be a regular character.                        */
                else{
                        input[0]=in;
                        input[1]='\0';
                        len=squeezein(line,input,&len,&ln,&maxln);
                       }
               }
        }
        return(1);
}

/* pro_html_line - just processes the menu functions for the HTML */
/*  options                                                       */
int pro_html_line(char *line, int last_br, int *maxln, int *len, int *ln)
{
        char *name;
        char temp[5];
        char input[2];
        char holder[50];

        int in;

        message("Enter a Selection: ");
        in=wgetch(meswin);
        if(in!=ERR){

/* Menu options first */
        if(in=='e' || in=='E') return(last_br);

/* Italicize Text */
        else if(in=='i' || in=='I'){
                if(italic_tag==0){
                        *len=squeezein(line,"<i>",len,ln,maxln);
                        italic_tag=1;
                        message("Choose 'Italics' again to close");
                       }
                else{
                        *len=squeezein(line,"</i>",len,ln,maxln);
                        italic_tag=0;
                        message("");
                       }
                return(0);
               }

/* Bold Text */
        else if(in=='b' || in=='B'){
                if(bold_tag==0){
                        *len=squeezein(line,"<b>",len,ln,maxln);
                        bold_tag=1;
                        message("Choose 'Bold' again to close");
                       }
                else{
                        *len=squeezein(line,"</b>",len,ln,maxln);
                        bold_tag=0;
                        message("");
                       }
                return(0);
               }
```

```
/* Center Text */
        else if(in=='c' || in=='C'){
                if(center_tag==0){
                        *len=squeezein(line,"<center>",len,ln,maxln);
                        center_tag=1;
                        message("Choose 'Center' again to close");
                       }
                else{
                        *len=squeezein(line,"</center>",len,ln,maxln);
                        center_tag=0;
                        message("");
                       }
                return(0);
               }


/* Otherwise format text */
        else if(in=='f' || in=='F'){
                if(font_tag==0){
                        *len=squeezein(line,"<font ",len,ln,maxln);
                        font_tag=1;
                        name=(char*)malloc(sizeof(char)*MAX_FSIZE);
                        name=get_name("Font Size? (1-7) (RETURN if no change):  ","",
                        if(name!=NULL){
                                in=atoi(name);
                                if(in>=1 && in<=7){
                                        sprintf(holder,"size=%d ",in);
                                        *len=squeezein(line,holder,len,ln,maxln);
                                       }
                               }
                        menu_bot_color();
                        *len=squeezein(line,"color=#",len,ln,maxln);
                        name=get_name("Color (6 digits, each 0-9 or A-F): ","",0);
                        menu_bot_edithtml();
                        sprintf(holder,"%s>",name);
                        *len=squeezein(line,holder,len,ln,maxln);
                        message("Choose 'Font' again to close");
                        free(name);
                       }
                else{
                        *len=squeezein(line,"</font>",len,ln,maxln);
                        font_tag=0;
                        message("");
                       }
                return(0);
               }

/* Add a Hyperlink */
        else if(in=='l' || in=='L'){
                if(link_tag==0){
                        temp[0]='"';
                        temp[1]='\0';
                        sprintf(holder,"<A HREF=%s",temp);
                        *len=squeezein(line,holder,len,ln,maxln);
                        name=(char*)malloc(sizeof(char)*MAX_FSIZE);
                        name=get_name("Address of the link?  ","http://",0);
                        temp[1]='>';
                        temp[2]='\0';
                        sprintf(holder,"%s%s",name,temp);
                        *len=squeezein(line,holder,len,ln,maxln);
                        link_tag=1;
                        message("Insert link text now, choose 'Hyperlink' again to cl
                       }
                else{
                        *len=squeezein(line,"</A>",len,ln,maxln);
```

```
                        link_tag=0;
                        message("");
                       }
                return(0);
               }

/* Add an Image */
        else if(in=='p' || in=='P'){
                temp[0]='"';
                temp[1]='\0';
                sprintf(holder,"<IMG SRC=%s",temp);
                *len=squeezein(line,holder,len,ln,maxln);
                name=(char*)malloc(sizeof(char)*MAX_FSIZE);
                name=get_name("Name of the Image File?  ","",0);
                temp[1]='>';
                temp[2]='\0';
                sprintf(holder,"%s%s",name,temp);
                *len=squeezein(line,holder,len,ln,maxln);
                message("");
                return(0);
               }

        else
                return(last_br);
       }
}
```

A:edmonson Job: menus.c Date: 1999-05-11-15:14:59

```
#include "main.h"

/*                                                                    */
/* This first set of menus controls what is displayed at the top */
/*  of the screen, the top window (duh)                               */
/*                                                                    */
void menu_top_init(){
        int i;

        wclear(topwin);
        wrefresh(topwin);
        wprintw(topwin,"UNIX Term HTML Editor - Brad Edmonson\n");
        for(i=1;i<COLS;i++)
                wprintw(topwin,"-");
        wrefresh(topwin);
}

void menu_top_editfile(char *name){
        int i;

        wclear(topwin);
        wrefresh(topwin);
        wprintw(topwin,"DOCUMENT:  %s\n",name);
        for(i=1;i<COLS;i++)
                wprintw(topwin,"-");
        wrefresh(topwin);
}

/*                                      */
/* These control the bottom window... */
/*                                      */
void menu_bot_init(){
        wclear(botwin);
        wrefresh(botwin);
        wprintw(botwin,"Enter a command");
        wrefresh(botwin);
}

void menu_bot_editfile(){
        wclear(botwin);
        wrefresh(botwin);
        wprintw(botwin,"^E: Exit(No Save)\t");
        wprintw(botwin,"^A: Save\t");
        wprintw(botwin,"^T: HTML Edit Options\t");
        wrefresh(botwin);
}

void menu_bot_edithtml(){
        wclear(botwin);
        wrefresh(botwin);
        wprintw(botwin,"E: End HTML Edit\t");
        wprintw(botwin,"L: Hyperlink\t");
        wprintw(botwin,"F: Font Options\t");
        wprintw(botwin,"I: Italics\t");
        wprintw(botwin,"B: Bold\t");
        wprintw(botwin,"P: Picture\t");
        wprintw(botwin,"C: Center\t");
        wrefresh(botwin);
}

void menu_bot_color(){
        wclear(botwin);
        wrefresh(botwin);
        wprintw(botwin,"EXAMPLES:\t");
        wprintw(botwin,"White-FFFFFF\t");
        wprintw(botwin,"Black-000000\t");
```

```
        wprintw(botwin,"Red-FF0000\t");
        wprintw(botwin,"Green-00FF00\t");
        wprintw(botwin,"Blue-0000FF\t");
        wprintw(botwin,"Yellow-FFFF00\t");
        wprintw(botwin,"Magenta-FF00FF\t");
        wprintw(botwin,"Cyan-0000FF\t");
        wrefresh(botwin);
}

/*                                          */
/* These control the main, central, window */
/*                                          */
void menu_cen_init(){
        wclear(cenwin);
        wrefresh(cenwin);
        wprintw(cenwin,"\n\tN - New HTML Document\n\n");
        wprintw(cenwin,"\tO - Open HTML Document\n\n");
        wprintw(cenwin,"\tX - Exit Editor\n\n");
        wprintw(cenwin,"\tH - Help\n\n");
        wrefresh(cenwin);
}

/*                                                  */
/* These are just some "other" window-text things */
/*                                                  */
void put_cursor_here(WINDOW *win){
        wrefresh(win);
        wprintw(win,"");
        wrefresh(win);
}
```

A:edmonson Job: other.c Date: 1999-05-11-15:15:11

```
/* These are other functions that didn't fit well anywhere else */

#include "main.h"

/* goodbye_cruel_world - egress function */
void goodbye_cruel_world()
{
        endwin();
        printf("\n");
        exit(1);
}

/* get_name - Takes three inputs:  prompt (the prompt used to */
/*  prompt the user for the information you would like),      */
/*  start (if you would like to initialize the return value   */
/*  to anything), and flag (0 leaves the cursor at the        */
/*  beginning of start while 1 leaves it at the end).         */
/*                                                            */
/*  Returns a character string input by the user or NULL if   */
/*  nothing is inputted or there is an error.                 */
char *get_name(char *prompt, char *start, int flag)
{
        int in,val;
        char *name;

/* Print the prompt and initialization */
        wclear(meswin);
        wprintw(meswin,prompt);
        wprintw(meswin,start);
        wrefresh(meswin);

/* Copy start into our return variable, it can be overwritten */
        name=strdup(start);
        val=strlen(start);

/* Move to the end of the input if desired */
        if(flag==1)
                while(val>0){
                        wmoveprevch(meswin);
                        val--;
                       }
        wrefresh(meswin);

/* Get the input, one character at a time */
        in=wgetch(meswin);
        while(1){
                while(in!=ERR && val<MAX_FSIZE-1){
                        if(in=='\n' && val!=0){
                                name[strlen(name)]='\0';
                                return(name);
                               }
                        else if(in=='\n' && strlen(name)==0)
                                return(NULL);
                        else if(in=='\n' && strlen(name)>0){
                                name[strlen(name)]='\0';
                                return(name);
                               }
                        else if(in==KEY_LEFT){
                                if(val>0){
                                        wmoveprevch(meswin);
                                        val--;
                                       }
                               }
                        else if(in==KEY_RIGHT){
                                wmovenextch(meswin);
                                val++;
```

```
                        }
                    else{
                            name[val]=in;
                            val++;
                            wprintw(meswin,"%c",in);
                        }
                    in=wgetch(meswin);
                   }
            if(val>MAX_FSIZE-1)
                    return(NULL);
           }
    return(NULL);
}

/* message - simplifies the procedure for printing to the message */
/*  window                                                        */
void message(char *output)
{
    wclear(meswin);
    wrefresh(meswin);
    wprintw(meswin,output);
    wrefresh(meswin);
    put_cursor_here(cenwin);
}

/* copy_to_file - copies the data structure into the given file */
void copy_to_file(Dlist lineset, int maxln, FILE *file)
{
    Dlist tmp;

    dl_traverse(tmp,lineset){
            fprintf(file,"%s",tmp->val);
           }
}

/* line_right - returns a string that is everything to the right */
/*  of the current location                                      */
char *line_right(char *line, int loc)
{
    int i,j;
    char *partline;

    partline=(char*)malloc((sizeof(char)*COLS)+100);

    j=0;
    for(i=loc;i<strlen(line);i++){
            partline[j]=line[i];
            j++;
           }

    partline[j]='\0';

    return(partline);
}

/* to_the_right - shifts a line to the right gap spaces */
void to_the_right(char *line, int loc, int gap)
{
    int i;
    char *oldline;

    oldline=strdup(line);

    for(i=loc;i<strlen(oldline);i++)
            line[i+gap]=oldline[i];
```

```
        line[i+gap]='\0';
}

/* blackout - backspaces both on the screen and in the data structure */
int blackout(char *line, int loc)
{
        int i;
        char *oldline,*partline;

        if(loc!=0){
                oldline=strdup(line);

                for(i=strlen(oldline);i>=loc;i--)
                        line[i-1]=oldline[i];

                wmoveprevch(cenwin);
                wdelch(cenwin);
                wrefresh(cenwin);

                return(loc-1);
               }

        else
                return(loc);
}

/* squeezein - reads input into the data structure and onto the */
/*  screen, also shifts stuff to the right if necessary         */
int squeezein(char *line, char *input, int *loc, int *ln, int *maxln)
{
        int i,j,k;
        char *partline;

        partline=strdup(line_right(line,*loc));
        spillcheck(strlen(input),line,loc,ln,maxln);
        to_the_right(line,*loc,strlen(input));
        for(i=0;i<strlen(input);i++){
                line[*loc+i]=input[i];
                winsch(cenwin,input[i]);
                wmovenextch(cenwin);
               }
        wrefresh(cenwin);
        return(*loc+strlen(input));
}

/* spillcheck - checks to see if we're about to go off the side of */
/*  the screen and creates a new line if we are                    */
void spillcheck(int size, char *line, int *len, int *ln, int *maxln)
{
        if((size+strlen(line))>(COLS-BUFFER)){
                wprintw(cenwin,"\n");
                free(cline->val);
                cline->val=strdup(line);
                dl_insert_b(lineset,"");
                cline=cline->flink;
                *len=0;
                *ln++;
                *maxln++;
                line[0]='\0';
               }
}
```

A:edmonson Job: fields.c Date: 1999-05-11-15:11:57

```
#include <stdio.h>
#include "fields.h"

#define talloc(ty, sz) (ty *) malloc (sz * sizeof(ty))
#define strdup(s) ((char *) strcpy(talloc(char, strlen(s)+1), s))

static IS make_inputstruct(filename, key)
char *filename;
char *key;                /* "f" for regular file or stdin if filename is NULL */
                          /* "p" if filename is a command for popen */
{
  IS is;
  int file;

  if (strcmp(key, "f") == 0) {
    file = 1;
  } else if (strcmp(key, "p") == 0) {
    file = 0;
  } else {
    return NULL;
  }

  is = talloc(struct inputstruct, 1);

  is->text1[MAXLEN-1] = '\0';
  is->NF = 0;
  is->line = 0;
  if (filename == NULL) {
    is->name = "stdin";
    is->f = stdin;
  } else {
    is->name = filename;
    is->file = file;
    if (file) {
      is->f = fopen(filename, "r");
    } else {
      is->f = popen(filename, "r");
    }
    if (is->f == NULL) {
      free(is);
      return NULL;
    }
  }
  return is;
}



IS new_inputstruct(filename)   /* use NULL for stdin.  Calls malloc */
char *filename;
{
  return make_inputstruct(filename, "f");
}

IS pipe_inputstruct(command)
char *command;
{
  return make_inputstruct(command, "p");
}

int get_line(is)
IS is;
{
  int i, len;
  int f;
  char *tmp;
```

```
  char lastchar;
  char *line;

  is->NF = 0;

  if (fgets(is->text1, MAXLEN-1, is->f) == NULL) {
    is->NF = -1;
    return -1;
  }

  is->line++;
  strcpy(is->text2, is->text1);

  line = is->text2;
  lastchar = ' ';
  for (i = 0; line[i] != '\0' && i < MAXLEN-1; i++) {
    if (isspace(line[i])) {
      lastchar = line[i];
      line[i] = '\0';
    } else {
      if (isspace(lastchar)) {
        is->fields[is->NF] = line+i;
        is->NF++;
      }
      lastchar = line[i];
    }
  }
  return is->NF;
}

void jettison_inputstruct(is)
IS is;
{
  if (is->f != stdin) {
    if (is->file) {
      fclose(is->f);
    } else {
      pclose(is->f);
    }
  }
  free(is);
  return;
}
```

A:edmonson Job: fields.h Date: 1999-05-11-15:12:48

```
#define MAXLEN 1001
#define MAXFIELDS 1000

typedef struct inputstruct {
  char *name;                 /* File name */
  FILE *f;                    /* File descriptor */
  int line;                   /* Line number */
  char text1[MAXLEN];         /* The line */
  char text2[MAXLEN];         /* Working -- contains fields */
  int NF;                     /* Number of fields */
  char *fields[MAXFIELDS];    /* Pointers to fields */
  int file;                   /* 1 for file, 0 for popen */
} *IS;

extern IS new_inputstruct(/* FILENAME -- NULL for stdin */);
extern IS pipe_inputstruct(/* COMMAND -- NULL for stdin */);
extern int get_line(/* IS */); /* returns NF, or -1 on EOF.  Does not
                                  close the file */
extern void jettison_inputstruct(/* IS */);  /* frees the IS and fcloses
                                                the file */
```

A:edmonson Job: dlist.c Date: 1999-05-11-15:12:57

```
/*
 * $Source: /n/fs/vd/jsp/src/jgraph/RCS/list.c,v $
 * $Revision: 5.4 $
 * $Date: 91/06/06 16:14:15 $
 * $Author: jsp $
 */

#include <stdio.h>    /* Basic includes and definitions */
#include "dlist.h"

#define boolean int
#define TRUE 1
#define FALSE 0


/*-------------------------------------------------------------------------*
 * PROCEDURES FOR MANIPULATING DOUBLY LINKED LISTS
 * Each list contains a sentinal node, so that
 * the first item in list l is l->flink.  If l is
 * empty, then l->flink = l->blink = l.
 *-------------------------------------------------------------------------*/

Dlist make_dl()
{
  Dlist d;

  d = (Dlist) malloc (sizeof(struct dlist));
  d->flink = d;
  d->blink = d;
  d->val = (void *) 0;
  return d;
}

dl_insert_b(node, val)  /* Inserts to the end of a list */
Dlist node;
void *val;
{
  Dlist last_node, new;

  new = (Dlist) malloc (sizeof(struct dlist));
  new->val = val;

  last_node = node->blink;

  node->blink = new;
  last_node->flink = new;
  new->blink = last_node;
  new->flink = node;
}

dl_insert_list_b(node, list_to_insert)
Dlist node;
Dlist list_to_insert;
{
  Dlist last_node, f, l;

  if (dl_empty(list_to_insert)) {
    free(list_to_insert);
    return;
  }
  f = list_to_insert->flink;
  l = list_to_insert->blink;
  last_node = node->blink;

  node->blink = l;
  last_node->flink = f;
```

```
  f->blink = last_node;
  l->flink = node;
  free(list_to_insert);
}

dl_delete_node(item)              /* Deletes an arbitrary iterm */
Dlist item;
{
  item->flink->blink = item->blink;
  item->blink->flink = item->flink;
  free(item);
}

dl_delete_list(l)
Dlist l;
{
  Dlist d, next_node;

  d = l->flink;
  while(d != l) {
    next_node = d->flink;
    free(d);
    d = next_node;
  }
  free(d);
}

void *dl_val(l)
Dlist l;
{
  return l->val;
}
```

A:edmonson Job: dlist.h Date: 1999-05-11-15:13:12

```
typedef struct dlist {
  struct dlist *flink;
  struct dlist *blink;
  void *val;
} *Dlist;

/* Nil, first, next, and prev are macro expansions for list traversal
 * primitives. */

#define dl_nil(l) (l)

#define dl_first(l) (l->flink)

#define dl_last(l) (l->blink)

#define dl_next(n) (n->flink)

#define dl_prev(n) (n->blink)

/* These are the routines for manipluating lists */

extern Dlist make_dl();
extern dl_insert_b(/* node, val */); /* Makes a new node, and inserts it before
                                        the given node -- if that node is the
                                        head of the list, the new node is
                                        inserted at the end of the list */
#define dl_insert_a(n, val) dl_insert_b(n->flink, val)

extern dl_delete_node(/* node */);    /* Deletes and free's a node */

extern dl_delete_list(/* head_node */);  /* Deletes the entire list from
                                            existance */
extern void *dl_val(/* node */);   /* Returns node->val (used to shut lint
                                      up) */

#define dl_traverse(ptr, list) \
  for (ptr = dl_first(list); ptr != dl_nil(list); ptr = dl_next(ptr))
#define dl_empty(list) (list->flink == list)
```