Database Maintenance for a Video Editing System

by

Andria H. Wong

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science in Electrical Science and Engineering
at the
Massachusetts Institute of Technology
May 1986

Signature of Author _____
Department of Electrical Engineering and Computer Science
May 9, 1986

Certified by _____
Glorianna Davenport
Thesis Supervisor

Accepted by _____
David Adler
Chairman, Department Committee

ABSTRACT

This thesis works with the data maintenance mode of a video editing system. The editing system we are concerned with consists of three modes : the Maintenance Mode, the Preview Mode, and the Execute Mode. The Maintenance Mode takes care of the video database, namely the edit decision list; the Preview Mode allows one to preview the result of edit decisions, and finally the Execute Mode does the actually editing. Data maintenance is done entirely in software. Utility programs are written to retrieve and modify information in the database, display sequences of edit decisions, and interact with other modes.

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my thesis supervisor, Glorianna Davenport, for allowing me to participate in this project of the film/video group at the MIT Media Laboratory. Many thanks also go to project directors Keishi Kandori and Russ Sasnett for their technical guidance along the way. I also wish to show my appreciation to all other members in the group for their efforts in making this project possible.

I am indebted to my parents for their love and financial support throughout my four years of college, and to my brothers and sisters for their encouragement as I undergo various difficulties and pressure.

Finally, I wish to express my deepest gratitude to the Father, whom I love dearly, and who has been helping me in every way he could to keep me growing in his love and mercy.

Table of Contents

Chapter 1

OVERVIEW

1.1 Introduction

Since the advent of computer-controllable video machines, TV post-production has gained much attention and has become very important in the television industry. One aspect of video production that has undergone tremendous progress is video-editing and post-production. Video-editing is no more the crude "cut and splice" of earlier days; it has become a sophisticated computerized craft, in which the entire post-production process is performed in a computerized editing bay. All the information that describes how the raw footage is to be transformed into a polished TV program is maintained in a database. This thesis describes how the information is represented, and explores ways to improve the editing process by making it easy for a human editor to create such a data base.

1.2 Data Maintained For Video Editing

There are two parts in the process of video editing. They are offline editing and online editing. They both maintain information in one of a number of similarly organized formats which are represented as Edit Decision Lists (EDLs). The offline editing process is when the editor has to choose the preferred takes from all the footage and to figure out how these sources are arranged to give a desirable output. This process is usually done in a small inexpensive offline system, after which the editor enters the offline edit decisions by hand. These offline EDLs have a less restricted format that suit the taste of the individual editor. It may only require the time code of the "edit-in" point and the scene duration as far as the location of the source is concerned. It may also include some verbal description of the take as well. The online editing process is done in a sophisticated computerized editing bay, where all the information in the EDLs must be clearly specified. This is usually a rather expensive process due to the hardware involved and its requiring the presence of a trained operator. Recently the distinction between offline and online editing becomes less obvious because some offline systems have been developed to such sophistication that an online EDL may be generated in the offline process. In general, the online EDLs include all the data listed below.

- Edit numbers for each shot, numbered consecutively.
- A reel number for each shot.
- The edit mode (audio-only, video-only, or audio and video) plus the number of the audio track on which audio edits will be recorded.
- The transition type (cut, dissolve, wipe, key etc).
- The time code of the edit start point and end point.
- The time code of the record start point and end point.

For the rest of this paper, we will only look at the online EDLs and its details will be discussed at length in the following chapters.

1.3 SMPTE/EBU Time Code

One major progress in the video industry is the advent of SMPTE/EBU (Society of Motion Picture and Television Engineers / European Broadcasting Union) time code. This is a standard format, set up by the SMPTE Committee, in which each individual frame address is to be represented. A SMPTE time code is a string of binary signals recorded along an audio track of a video tape or in the vertical interval between two fields which make up a video frame so that each frame or field has a unique code. As far as the time code in the EDL is concerned, it is represented as a string of eight digits. SMPTE time code is indexed in hours, minutes, seconds and

frames with 30 frames per second in NTSC (National Television Standards Committee).

10 : 23 : 42 : 12

HR : MIN : SEC : FR

NTSC time code readouts range from 00:00:00:00 to 23:59:59:29 , recycling at each 24-hour interval.

SMPTE time code is essential to computerized video editing for the following reasons :

1. Accuracy and repeatability. Time code permits edits that are accurate down to one frame or field, and that are repeatable. Edit points can also be adjusted in one frame increments.

2. Precise time reference. The duration of a selected scene can be determined with frame accuracy, simply by subtracting the "edit-in" point from the "edit-out" point. Time code also gives editors frame-accurate running times throughout the editing process.

3. Interchangeability between editing systems. Editors can perform their offline editing on a low-cost equipment and then input the results to a high quality online editing system. This is a money saving strategy.

4. Precision synchronization of one VTR to another. Time code allows an editing system to bring two or more tapes into exact sync automatically. This means perfect frame-to-frame match-ups at the edit points with no video breakups.

Video editing industry owes a major part of its advancement to the standardization of this frame indexing technique. In a later chapter, we shall look at how SMPTE time code is utilized in the EDL database.

Chapter 2

EDL CLOSE-UP

2.1 Introduction

Although SMPTE has formed a committee to work on establishing a standardized industry-wide format for computerized edit decision lists, no single standard exists today. The most widely used EDL format is the ASCII format (American Standard Code for Information Interchange) or more commonly referred to as the CMX format.

This chapter describes the structure of this format, the details of its data fields, and discusses the EDL limitations in our video editing system.

2.2 CMX Format And Its Data Fields

The CMX format of EDL consists of ten discrete areas of information or data fields.

1. Field 1 contains the edit event number. This indicates the numerical order by which the edits in the list will be performed. In the case of a two-line edit, the two lines of information should both contain the same event number.

2. Field 2 indicates the source of the edit. This may be a reel number or abbreviations such as "AX" for auxiliary, "BL" for black or "ISO" for isolated camera.

3. Field 3 indicates the edit mode. This may include : "AV", designation for audio/video; "A1" or "A2", designatin for audio-only plus the specification of the audio track ; "V", designation for video-only edits.

4. Field 4 designates the edit transition type. This may include any of the following letter/number combinations :

   - C
     CUT

   - D
     DISSOLVE transition. This is a two-line edit with the first line indicating the outgoing scene.

   - W***
     WIPE transition. This is a two-line edit. The three asterisks stand for a three digit wipe code specifying which wipe configuration the video switcher should perform.

   - KB
     KEY transition. The B indicates that it is the first line of a two-line edit and it is the background source.

   - K
     KEY-IN transition. This is the second line of a two-line edit and it is the foreground source.

   - KO
     KEY-OUT transition. This is the second line of a two-line edit, and it is the foreground source that will be on when the edit starts and eventually fades out after a prescribed duration.

5. Field 5 is a three digit number in frames indicating the duration at which the transition between the two sources of a two-line edit should take place. It is blank for a one-line edit, namely a CUT.

6. Field 6 contains the start time of the playback VTR, designated in SMPTE time code.

7. Field 7 contains the stop time of the playback VTR, designated in SMPTE time code.

8. Field 8 contains the start time of the record VTR, designated in SMPTE time code.

9. Field 9 may either contain the stop time of the record VTR or the duration of the edit.

10. Field 10 contains the carriage-return which is not actually printed in the edit decision list readout.

2.3 Our EDL Limitations

There are a number of differences between the CMX format and the EDLs supported by our editing system. Some of these are considered limitations because our scheme for data maintenance has not been developed to such sophistication.

- Field 2 :
A reel number is represented by a number instead of a short character string, and we do not support "ISO" (isolated camera).

- Field 3 :
We use "B" , which stands for "Both", instead of "AV" to designate an audio/video edit.

- Field 4 :
Instead of having "KB" and "KO", "K" by itself stands for a key transition with the first line specifying the background source, and the second line, the foreground source.

- Field 5 :
Instead of a three digit frame count of transition duration, we use the format of 02:08 to indicate a duration of two seconds and eight frames. For the convenience of reading EDL, we append the duration of 00:00 in place of the blanks.

- Field 9 :

This field always contains the SMPTE "edit-out" point for the stop time of the record VTR.

Aside from these limitations, our EDLs follow the CMX specifications. However, many features need to be added to our database management to take full advantage of the computational environment. Some possible expansions will be discussed in chapter 4.

Chapter 3

IN PREPARATION FOR EDL ENTRY UTILITY

3.1 Introduction

In preparation for the EDL entry utility, two other programs were written. They are respectively MakeEDL and Talk. All software is written in C and run in a UNIX FIVE environment on a Hewlett-Packard Bobcat. This chapter describes in detail these two programs and discusses their limitations and ways in which they could be improved.

3.2 MakeEDL

MakeEDL creates a list of edit decisions and saves it in a file called "sample.edl". This is a straight-forward program that generates 512 edit decisions solely to be experimented upon by the next two programs, Talk and Edlentry. The information in the list is largely "dummy" data, containing no significant meaning at all. Nevertheless ,it does demonstrate all of the various types of edits, which a real list might not provide. This is important for error-checking.

The EDL begins with the playback and record start times at 1 hour (01:00:00:00). Each subsequent event

line of the edit list has a duration of 10 frames. This is done by incrementing the integer array newtc by 10 frames every time makenewtc is called.

Events 1 to 5 are each two-line edits with audio/video mode and a dissolve transition.

Events 6 to 9 are also two-line edits in audio/video mode with a wipe transition of the 101 wipe configuration.

Events 10 to 99 are all CUTs with the edit mode of audio track 2.

Events 100 to 512 are also CUTs with video-only edit mode.

The source code of MakeEDL can be found in Appendix A and a simplified EDL sample of 12 events is also appended for reference at the end of the program listing.

3.3 Talk

This program accesses the file "sample.edl" and reads or modifies the information it contains upon receiving an appropriate command from the terminal. Talk starts with an infinite loop by busy-waiting for a request typed by an operator at the keyboard. It accepts an input string, and tries to match it against each of the three valid commands. If there is no match, it gives an error message and returns to the waiting mode.

The valid commands are : SE#*, RE1#* and RE2#* ('*' stands for an event number).

SE#* is a command for reading a specified event. Talk will test for the validity of the event number, which must be between 1 and 512 inclusive. If it is valid, Talk searches the file "sample.edl" for the specified event number. Once it is found, the whole line of information appended to the string "be#*" is printed on the terminal (2 lines in the case of a two-line edit).

RE1#* and RE2#* are commands for modifying the EDL and are somewhat more complicated to deal with than SE#*, because the old event or the new event can be either a one-line or two-line edit. RE1 signifies that the new event consists of one line, while RE2 is that of two lines. After the event number has passed the validity test, we must check and see what category of replacement the request belongs to. There are altogether four categories. We call the replacement of a one-line edit by a two-line edit a "1->2 replacement". So the meanings for the categories of "1->1 replacement", "2->2 replacement" and "2->1 replacement" are obvious. The operations for 1->1 and 2->2 replacements are simple. Talk accepts the new event from the terminal and writes it in the file where the old event is to be replaced. In the case of a 2->1 replacement, the second line of the old event must be deleted, while in the case of a 1->2 replacement, the second line of the new event must be inserted. These operations are done in the routine

updatef in which a temporary file is created. Sample.edl is copied into this tempfile up to the point where deletion or insertion should take place. In the case of deletion, the unwanted line in sample.edl is skipped; while in the other case, the wanted line is added to tempfile. Then the rest of the EDL is copied until end-of-file is reached. Finally, a system call is executed to move the contents of tempfile into sample.edl.

One limitation of this program is that it assumes sample.edl is a complete file, in that it contains all edit events from 1 to 512. It will not function if the event to be accessed is missing. Nevertheless, this limitation can be removed by further checking the EDL and by giving appropriate operations.

Talk can be made to work more efficiently if instead of accessing the file for every valid request, it copies the whole file into an array and operates on the array alone. In such case, the 1->2 and 2->1 EDL replacements can be handled more easily and rapidly. Then upon exit of the program, sample.edl can be updated by simply writing the array back into the file. This method is used in the next program Edlentry.

The source code of Talk is listed in Appendix B, and we will proceed with Edlentry in the next chapter.

Chapter 4

EDL ENTRY UTILITY

4.1 Introduction

The major program written for database maintenance is Edlentry. This EDL entry utility allows the editor to create a new file of EDL or modify an existing one with ease. Since much attention has to be given to the aspect of human interface, the distinguished feature of window management supported by the HP-Bobcat seems especially helpful. Since the entire EDL is composed of text, no special graphic features are necessary. A window package designed for text called "curses" from the standard UNIX support is enough to provide a nice editing environment on the HP terminal. We will soon be able to appreciate the convenience it brings as we come to understand how it is being utilized. The details of Edlentry and its limitations will be discussed in the following sections.

4.2 Edlentry

Edlentry is a rather large and complex program. In order to keep it modular, I have the major routines called by the main program stored in separate files. This modular characteristic can be seen by studying the

main program in the file "edlentry.c". Since the window package "curses" will be used extensively many portions of the program, the file "curses.h", which makes the necessary declarations and which contains the standard C i/o header file "stdio.h", is included in almost all files. Common variables shared by different files are defined in "edldef.h" and declared in "edlentry.c". "locdef.h" is another header file that contains the definitions of EDL data locations on the menu window which will be explained later.

One systematic way to understand Edlentry is to walk through the main loop. The main loop is a simple program that calls a number of subroutines. The first four are curses functions that set up the terminal configuration such as clearing the screen. The next routine called is menudisp which opens a window (menuwin), using curses, on the terminal to display a menu. This menu consists of a number of entry items for information that make up an edit decision. This window acts as an editing screen when we are in the edit mode. The next routine is edlcmd which opens another window (cmdwin) at the bottom of the terminal and prompts the user for the EDL file to be accessed. If it is a new file, it is created. If it is an old file, it is read and copied into some memory allocated upon the calling of the next routine readedl. Readedl keeps an array of pointers edlptr[] to each line

of edits. The main loop then calls edldisp which opens a third window (edlwin) and places it between menuwin and cmdwin. Edldisp displays the first 20 lines of edits in the file designated by the user onto the edlwin. In the case of a new file, edlwin remains empty. The next routine called is edledit which is the most complex portion of the program and it deserves separate discussion. In short, it is where we enter the edit mode in an infinite loop. Within this mode, we can create new EDLs, delete unwanted EDLs or modify the old contents. Upon the exit of edledit, main calls the last function endwin which is a curses routine that destroys all windows created and sets the terminal back to the normal text mode.

4.3 EDL Edit Mode

At the moment we enter the routine edledit, we have the terminal divided into three windows. Menuwin contains the EDL entry items. Edlwin has a listing of 20 edit decisions. (It may be blank in the case of a newly created empty file.) Cmdwin shows the filename the user just entered, and this window is no longer in use. Inside the edit mode, all of the user's designated operations are done in the menuwin. Edlwin only reflects the changes the user has made in the EDL buffer. Therefore, this window must be updated whenever there is

a modification or an addition in the database. The first job edledit does is to read the first event from the EDL buffer, break it down into pieces of information and place them in their respective designated locations (defined in "locdef.h") on the menuwin. Duration of each edit is also calculated and displayed. (This procedure is skipped in the case of a new file.)

Edledit has now entered an infinite loop where it is waiting indefinitely for the user to enter a command. Once a valid command is received, appropriate procedures will be called. Usual operations requested by the user may be displaying a selected event on the menuwin, changing the information in the edit, saving the modified event, or just entering new events to build up a list of edit decisions, saving it on hard disk etc.

Specific commands that edledit accepts are the following :

A) Edit mode selections :

1. 'y' ----- audio/video
2. 'u' ----- audio 1 or audio 2
3. 'i' ----- video-only

When any of these keys is pressed, the parameter on the menuwin takes the appropriate value automatically. In the case of audio-only, the user is prompted for select-ion of audio track at the bottom of the menuwin.

B) Transition type selection :

1. 'p' ----- CUT
2. '[' ----- DISSOLVE
3. ']' ----- WIPE
4. '\' ----- KEY

When any of these keys is pressd, the parameter on the menuwin takes the appropriate value automatically. In the case of a WIPE, the user is prompted for a wipe code.

C) VTR selections :

1. 'a' ----- R-VTR
2. 's' ----- A-VTR
3. 'd' ----- B-VTR
4. 'f' ----- C-VTR
5. 'g' ----- D-VTR
6. 'h' ----- E-VTR
7. 'j' ----- F-VTR
8. 'k' ----- AUX
9. 'l' ----- BLK

When any of these keys is pressed, the cursor is moved to the appropriate location of the menuwin, and its y-coordinate is remembered. The default position of the cursor is at R-VTR.

D) Time code inputs :

1. 'x' ----- SET IN (set start time tc)
2. 'c' ----- SET OUT (set stop time tc)
3. 'z' ----- DURATION (set duration of edit)
4. 'v' ----- TRIM IN (modify present start time tc by an offset)
5. 'b' ----- TRIM OUT (modify present stop time tc by an offset)

When either SET IN or SET OUT is requested, the user is prompted for a SMPTE time code which can be entered with or without colons. Since the user has to select a VTR in advance, edledit knows where the code belongs.

When either DURATION or TRIM IN or TRIM OUT is requested, the user is prompted for a duration either in the form of "second:frame" or just the number of frames. In the case of a negative TRIM IN or a negative TRIM OUT, a minus sign must precede the duration input. When any time code is changed, a new duration is calculated and displayed. If the present data is invalid such as having time-in greater than time-out, the duration is left blank.

E) Other functions :

1. '8' ----- OPEN END. The time out and duration columns are erased.
2. '1' ----- Select event number for display. This is the same procedure as the first job done by edledit. If the event does not exist,

the event number is simply placed in the right entry location, and the user can start filling in informations.

3. CTRL(L) ----- SAVE EVENT. Check to see if the present data make a legal edit. Prompt for source 1, source 2 and a transition duration if it is a two-line edit. Save the event in the EDL buffer, update the edlwin and make menuwin ready for a new EDL input.

4. '.' ----- DELETE EVENT. Prompt the user for unwanted event number and remove it from the EDL buffer.

5. '6' ----- SAVE DATA ON DISK. Write the contents of the EDL buffer back on to the original file. Acknowledge when operation is over.

6. 'Q' ----- QUIT. Ask if the user wants to save the data on disk. (This prevents the user from exiting the program without updating the file.) Confirm the user's decision to quit.

For the commands listed above, most of the keys with their corresponding functions follow the CMX keyboard configuration. For those commands that prompt for inputs, a carriage-return aborts the operation. This saves the user from performing an unwanted operation upon mistakenly hitting a key. Edledit either prints out an error message when it receives an illegal input or just ignores it totally without echoing on the menuwin.

Edledit supports the most basic operation necessary for the Edlentry program. At this stage, I will point out some of its major limitations. A well-prepared EDL automatically updates its record time codes such that, as you go down the list, the time-out of one event is

exactly the time-in of the next event, and that event is now ready for actual video editing. In our case, after a SAVE EVENT or a DELETE EVENT operation is performed, the feature of record time code flow may have been disturbed. To solve this problem, we can write a procedure, ripple, to be called after a save or delete operation, which updates the record in/out time codes down the list, from the point where a disturbance is made, maintaining the specified duration of each edit but re-calculating the time codes of the record fields.

Another limitation is within the procedure SAVE EVENT. Since the duration of record time code must be equal to the duration of playback time (or sum of the playback durations in the case of a two-line edit). The information entered by the user is allowed to have a number of unknowns. A smart program can be made to calculate the unknowns from the data given. This involves much more checking and is another area the program can be expanded. So far, our progaram requires all information to be given for the source VTR (VTRs) as well as requiring the user to input the time code of the record start time. Lacking any of those the program will signal an error and will give up on the operation. The functions of the program are limited. No doubt that the program can be expanded much further, and some suggestions are presented in the next section.

A simplified table of keys and their functions and the source code of Edlentry can be found in Appendix C.

4.4 Future Expansions For Edlentry

In addition to the suggested modifications of the functions already implemented, there are many features one can introduce to the Edlentry program.

Some recommended expansions are the following :

1. Edit mode designation
   Since multiple-audio-track VTRs are becoming common in video post-production, the EDL field 3 may be expanded to include a combination of video and multiple audio track designations. For instance, V12 may represent an edit for recording video and audio tracks 1 and 2.

2. Format for entering time code
   The hour and minute fields of a record time code usually remains the same for a number of consecutive edits. The editor might get tired of punching in the same sequence after a while. It may be helpful to implement a feature that allows the user to store such a repeated sequence in register and only enter the second and frame fields as inputs to functions that alter time codes.

3. Record time sorting
   Our Edlentry program assumes all edits are entered by the consecutive ascending order of their event numbers (field 1). It would be a problem if we want to insert an edit between two consecutively numbered events. A record time sorting function can be implemented to rearrange the list so that the edits are ordered by their record time. Since the event lookup routine, findev, called by many functions, require the event numbers to be in an ascending order, this record time sorting function should also be made to fix the event numbers so that the list still meets the requirement for findev.

4. Record showtime
   A parameter displayed on the menuwin indicates the duration of recording since the beginning of all edits. This is just the sum of edit durations until the present moment.

5. Event highlighting
   The particular event which the editor is working on in the edit mode can be highlighted on the edlwin so that it stands out and looks good on the terminal screen. Curses provides commands that do the highlighting.

6. Direct change of data on menuwin
   All implemented functions that require data from the user prompt for the input on a special input line at the bottom of the menu window, and updates the menuwin upon receiving valid data. Another way to implement this is to move the cursor to the right position where the corresponding command is issued allowing for direct input at the point so that no updating is necessary.

7. EDL format variations
   In the future, the EDL format may need to be expanded as to include data to set up and activate digital video effects devices, to identify special features of video and audio switching equipment and to control variable speed motion features.

There are certainly many more ways one can improve the EDL entry utility. The scope of possible expansion continues to grow as the video production technology continues to develop.

Chapter 5

CONCLUSION

5.1 Standardization

The Society of Motion Picture and Television Engineers (SMPTE) has formed a number of study groups to work on the standardization of various hardware and software features in video production and post-production processes. One example is the SMPTE time code which has been successfully established. The purpose of standardization is to keep individual components compatible with one another. A proposal for editing list standards has been drafted by the SMPTE committee on video recording and reproduction technology. As yet no single format has been adopted as universal.

5.2 Conclusion

There are a number of different things I have learned throughout this undergraduate thesis experience. Having no previous knowledge of the video world, this project seemed to be a tremendous challenge to undertake. My thanks go to the seminar, given by the film/video group last semester, which has provided me enough background and thus the confidence I need to plunge myself into working on this project.

Another precious experience has been working on the HP-Bobcat which is a relatively new system that just arrived at the Media-Lab this semester. Despite a number of bugs it has, the system supports some features that make it very convenient for writing software. Its window system allows one to work on a few files on the terminal at the same time. The sub-package "curses" offers a lot of routines necessary for the Edlentry program. The VI editor is fast and is easy to use. The fact that it uses the UNIX operating system gives me the chance to practice programming and become proficient in C.

One last thing I would like to mention is the opportunity, that I have gained from being a member of this experimental film/video group, to attend the NAB show (National Association of Broadcasters) held in Dallas, Texas, in April 1986. Even though the many audio, video systems presented by a large number of companies really seemed overwhelming, film/video remains a very fascinating and a challenging world which I may want to re-experience in the future.

APPENDIX A

```
/* makeEDL.c */
/* This program creates a list of dummy edit decisions and  */
   /*   saves it in the file "sample.edl"    */

#include <stdio.h>
#define CAT2SPACES      fprintf(fp,"  ")

FILE *fopen(), *fp;
int oldtc[4], newtc[4];

main()
{
        int i, hr, min, sec, fr;  /* variables that make up the SMPTE tc */
         char *filename;

filename = "sample.edl";
if ( (fp = fopen(filename, "w")) == NULL ) {
printf("can't open %s.\n", filename);
}

hr = 1;
min = sec = fr = 0;
newtc[0] = hr;
newtc[1] = min;
newtc[2] = sec;
newtc[3] = fr;

/* event 1 to 5 are 2-line edits with a DISSOLVE transition */
for (i = 1; i < 6; i++) {
araycpy(oldtc, newtc);
makenewtc(newtc);
fprintf(fp, "00%d  00%d  B    C     00:00  ", i, i);
tcput1();
araycpy(oldtc,newtc);
makenewtc(newtc);
fprintf(fp, "00%d  00%d  B    D     00:29  ", i, i+1);
tcput1();
}

/* event 6 to 9 are 2-line edits with a WIPE transition */
for (i=6; i<10; i++) {
araycpy(oldtc, newtc);
makenewtc(newtc);
fprintf(fp, "00%d  00%d  B    C     00:00  ", i, i/2);
tcput1();
araycpy(oldtc, newtc);
makenewtc(newtc);
fprintf(fp, "00%d  00%d  B    W101  00:29  ", i, i/2+1);
tcput1();
}

/* event 10 to 99 are all CUT's with edit mode of audio track 2 */
for (i = 10; i < 100; i++) {
araycpy(oldtc, newtc);
makenewtc(newtc);
fprintf(fp, "0%d  003  A2   C     00:00  ", i);
tcput1();
}
```

```
/* event 100 to 512 are one-line edits with video-only edit mode */
for (i = 100; i < 513; i++) {
araycpy(oldtc, newtc);
makenewtc(newtc);
fprintf(fp, "%d  005  V    C     00:00  ", i);
tcput1();
}

fclose(fp);
}


/* print the playback and record time codes by calling tcput2 */

tcput1()
{
        tcput2(fp, oldtc);
        CAT2SPACES;
        tcput2(fp, newtc);
        CAT2SPACES;
        tcput2(fp, oldtc);
        CAT2SPACES;
        tcput2(fp, newtc);
        fprintf(fp, "\n");
}

/* print a time code in the right format */

tcput2(fptr, tc)
        FILE *fptr;
        int tc[];
{
        int i;

for (i = 0; i < 3; i++) {
        if (tc[i] < 10) fprintf(fptr, "0");
        fprintf(fptr, "%d", tc[i]);
        fprintf(fptr, ":");
        }
if (tc[3] < 10) fprintf(fptr, "0");
        fprintf(fptr, "%d", tc[3]);
}


/* copy array newtc to oldtc */

araycpy(oldtc, newtc)
        int oldtc[], newtc[];
{
        int i;
        for (i=0; i<4; i++)
                oldtc[i] = newtc[i];
}


/* create a newtc by an increment of 10 frames */

makenewtc(tc)
        int *tc;
{
        int hr, min, sec, fr;
        hr = tc[0];
        min = tc[1];
        sec = tc[2];
```

```
        fr = tc[3];

        fr += 10;
        if (fr == 30) {
                sec++;
                fr = 0;
        }
        if (sec == 60) {
                min++;
                sec = 0;
        }
        if (min == 60) {
                hr++;
                min = 0;
        }
        tc[0] = hr;
        tc[1] = min;
        tc[2] = sec;
        tc[3] = fr;
}
```

```
/* Here is an example of EDL that is generated by a simplified
version of makeEDL.c.        */

001  001  B    C     00:00  01:00:00:00  01:00:00:10  01:00:00:00  01:00:00:10
001  002  B    D     00:29  01:00:00:10  01:00:00:20  01:00:00:10  01:00:00:20
002  002  B    C     00:00  01:00:00:20  01:00:01:00  01:00:00:20  01:00:01:00
002  003  B    D     00:29  01:00:01:00  01:00:01:10  01:00:01:00  01:00:01:10
003  003  B    C     00:00  01:00:01:10  01:00:01:20  01:00:01:10  01:00:01:20
003  004  B    D     00:29  01:00:01:20  01:00:02:00  01:00:01:20  01:00:02:00
004  004  B    C     00:00  01:00:02:00  01:00:02:10  01:00:02:00  01:00:02:10
004  005  B    D     00:29  01:00:02:10  01:00:02:20  01:00:02:10  01:00:02:20
005  005  B    C     00:00  01:00:02:20  01:00:03:00  01:00:02:20  01:00:03:00
005  006  B    D     00:29  01:00:03:00  01:00:03:10  01:00:03:00  01:00:03:10
006  003  A2   C     00:00  01:00:03:10  01:00:03:20  01:00:03:10  01:00:03:20
007  003  A2   C     00:00  01:00:03:20  01:00:04:00  01:00:03:20  01:00:04:00
008  004  A2   C     00:00  01:00:04:00  01:00:04:10  01:00:04:00  01:00:04:10
009  004  A2   C     00:00  01:00:04:10  01:00:04:20  01:00:04:10  01:00:04:20
010  005  A2   C     00:00  01:00:04:20  01:00:05:00  01:00:04:20  01:00:05:00
011  005  A2   C     00:00  01:00:05:00  01:00:05:10  01:00:05:00  01:00:05:10
012  006  A2   C     00:00  01:00:05:10  01:00:05:20  01:00:05:10  01:00:05:20
```

APPENDIX B

```
/* talk.c */
/* This program runs an infinite loop, it prints or modifies
        EDL's in the file "sample.edl" upon request on the console. */

#include <stdio.h>
#define EDLLEN  79      /* EDL length */
#define BEGIN   0               /* selected origin    */
#define HERE    1               /*   for the function */
#define END     2               /*      lseek         */
#define RE1     1       /* number of lines    */
#define RE2     2       /* for replacement    */
#define RWMODE  0666            /* read & write mode */
#define Eof     0

        char *filename;
        int evnum, curev, fd, evln;
        char buf[85];
        long offset, accpos;
        char str[10];
        char cmd[20];


/* start an infinite loop by busy waiting for command from terminal */
main()
{
        filename = "sample.edl";

        while(gets(str)) {
                /* SEnd event# : command for reading a specified event */
                if (sscanf(str, "se#%d", &evnum)== 1) {
                        rdedl();
                        continue;
                        }
                /* RE1 event# : replace a specified event by a 1-line edit */
                /* RE2 event# : replace a specified event by a 2-line edit */
                if (sscanf(str, "re%d#%d", &evln, &evnum)== 2) {
                        rpedl();
                        continue;
                        }
                printf("bad command\n");
                }
}


/* readedl : reads a specified event from "sample.edl"
        and prints it on the terminal.     */

rdedl()
{
        if (evnum==0 || evnum>512) {         /* test for validity of event # */
                printf("Event # %d does not exist.\n", evnum);
                return(1);
                }
        fd = open(filename,0);
        evlookup();                                 /* search for event */
        read(fd, buf, EDLLEN);
        printf("be#%d %s", evnum, buf);
        if (read(fd, buf, EDLLEN) != Eof) {
                sscanf(buf, "%d", &curev);
                if (curev == evnum)                 /* If it is a 2-line edit, */
```

```
                        printf("%s", buf);      /*  send the 2nd line. */
                }
        close(fd);
}


/* replace edl : replaces a specified event with the new EDL
        typed by an operator at the terminal */

rpedl()
{
        int n_read;

        if (evnum==0 || evnum>512) {        /* test for validity of event # */
                printf("Event # %d does not exist.\n", evnum);
                return(2);
                }
        fd = open(filename,2);
        evlookup();
        gets(buf);                          /* get new EDL from the terminal */
        strcat(buf,"\n");
        write(fd,buf,EDLLEN);
        accpos += EDLLEN;
        n_read = read(fd, buf, EDLLEN);
        if (n_read == Eof) {
                if (evln == RE2) {          /* replacing in the file the last  */
                        gets(buf);          /* event which is a 1-line edit by */
                        strcat(buf,"\n");       /* a 2-line edit */
                        write(fd, buf, EDLLEN);
                        }
                close(fd);
                return;
                }
        lseek(fd, -EDLLEN, HERE);
        sscanf(buf, "%d", &curev);
        if (evln == RE2) {
                gets(buf);
                strcat(buf,"\n");
                if (evnum == curev) {           /* 2 -> 2 replacement */
                        write(fd, buf, EDLLEN);
                        close(fd);
                        }
                else updatef();             /* 1 -> 2 replacement, 2nd line */
                }                           /* of new EDL must be inserted  */
        if (evln == RE1 && evnum == curev) updatef();  /* 2 -> 1 replacement */
        else close(fd);             /* 2nd line of old event must be deleted */
}


/* event lookup : searches for a specified event in the file,
        calculates its absolute position in the variable accpos.  */

evlookup()
{
        accpos = offset = (evnum - 1) * EDLLEN;
        lseek(fd, offset, BEGIN);
        read(fd, buf, EDLLEN);
        lseek(fd, -EDLLEN, HERE);
        sscanf(buf, "%d", &curev);
        while(curev != evnum) {
                offset = (evnum - curev) * EDLLEN;
                accpos += offset;
                lseek(fd, offset, HERE);
                read(fd, buf, EDLLEN);
                lseek(fd, -EDLLEN, HERE);
                sscanf(buf, "%d", &curev);
```

```
                }
}


/* update file : rewrites sample.edl due to 2->1 and 1->2 EDL replacement */
updatef()
{
        long curpos = 0;
        char *tempfile, tempbuf[85], *cmdbuf;
        int tempfd;

        tempfile = "tempfile";
        tempfd = creat(tempfile, RWMODE);       /* creates a temporary file */
        lseek(fd, 0, BEGIN);
        while (curpos != accpos) {             /* copy sample.edl to tempfile */
                read(fd, tempbuf, EDLLEN);      /* until the line where */
                curpos += EDLLEN;                /* replacement occurs */
                write(tempfd, tempbuf, EDLLEN);
                }
        if (evln==RE1 && evnum==curev)            /* 2->1 replacement */
                lseek(fd, EDLLEN, HERE);       /* skip 2nd line of old event */
        if (evln==RE2 && evnum!=curev)            /* 1->2 replacement */
                write(tempfd, buf, EDLLEN);  /* insert 2nd line of new event */
        while (read(fd, tempbuf, EDLLEN) != Eof)    /* copy the rest of the */
                write(tempfd, tempbuf, EDLLEN);  /* sample.edl to tempfile */
        close(fd);
        close(tempfd);
        cmdbuf = "mv ";
        strcpy(cmd,cmdbuf);
        strcat(cmd,tempfile);
        strcat(cmd," ");
        strcat(cmd,filename);
        system(cmd);          /* execute system call (mv tempfile sample.edl) */
}
```

APPENDIX C

```
/** keytbl---list of keys and their functions **/

/* AUDIO/VIDEO MODE */
'y'-----A/V (B)
'u'-----A1 or A2
'i'-----V

/* TRANSITION TYPE */
'p'-----CUT
'['-----DISSOLVE
']'-----WIPE
'\'-----KEY

/* VTR SELECTIONS */
'a'-----RVTR
's'-----AVTR
'd'-----BVTR
'f'-----CVTR
'g'-----DVTR
'h'-----EVTR
'j'-----FVTR
'k'-----AUX
'l'-----BLK

/* TIME CODE INPUTS */
'x'-----SET IN
'c'-----SET OUT
'z'-----DURATION
'v'-----TRIM IN
'b'-----TRIM OUT

/* OTHER FUNCTIONS */
'1'-----SELECT EVENT # FOR DISPLAY
'8'-----OPEN END
'.'-----DELETE EVENT
^L -----SAVE EVENT
'6'-----SAVE DATA ON DISK
'Q'-----QUIT
```

```
/*** locdef.h ---- location definition of parameters on menu window  ***/
/*** prefix 'X' stands for x-coordinate; 'Y' stands for y-coordinate ***/

/* column 1 */
#define XCOL1   2       /* column 1 */
#define YAVMODE 2       /* edit mode */
#define XAVMODE 5
#define YTRNMD  5       /* transition type */
#define YSRCDST 6       /* sources designation */
#define YTRNLEN 7       /* transition duration */
#define YEVNUM  9       /* event number */
#define XEVNUM  9
#define YINPUT  17      /* input line */

/* column 2 */
#define XVTR    15      /* VTR column */
#define YRVTR   4       /* record VTR */
#define YAVTR   6       /* A-VTR */

/* column 3 */
#define XIN     22      /* time in column */

/* column 4 */
#define XOUT    35      /* time out column */

/* column 5 */
#define XDUR    49      /* edit duration column */
```

```
/*  edldef.h ---- definitions of common variables and constants */

#define EDLLEN  79      /* EDL length */
#define Eof     0
#define TCLEN   11      /* time code length */
#define MAXEVNUM 512    /* biggest possible event number */
#define RWMODE  0666    /* read & write mode */
#define LBLANK  "                                          "        /* 38 spaces */
#define SHBLANK "            "          /* 12 spaces */
#define MTTC    "           "           /* 11 spaces */
#define CTRL(c) ('c'&037)

EXTERN  WINDOW *menuwin, *cmdwin, *edlwin;
EXTERN  char filename[10], *edlptr[1024];
EXTERN  int fd, lnstart, lnend, curln, tmptc[4];
EXTERN  int curev, num_of_ln, maxev, newfile, evnum;
EXTERN  int curry, currx, reel, oreel, wcode;
EXTERN  char avmo[3], tmode, tdur[5], sign;
EXTERN  char ptci[12], ptco[12], rtci[12], ortci[12];
EXTERN  char rtco[12], tcdur[12], evstr[3];
EXTERN  char rlstr[3], src[2], dst[2], tmdstr[5];
EXTERN  int ysrc, ydst;
```

```
/* edlentry.c ---- main program for EDL entry utility */
#include <curses.h>
#define EXTERN
#include "edldef.h"

main ()
{
        initscr();
        nonl();
        echo();
        cbreak();
        menudisp();     /* open a menu window and display */
        edlcmd();       /* open a command window and
                           prompt for file to be accessed */
        readedl();      /* read the EDL file */
        edldisp();      /* open an EDL window and display EDL */
        edledit();      /* edit mode */
        endwin();       /* destroy all windows and reset terminal */
        exit(0);
}
```

```
/* menudisp ---- create menuwin and display entry items */
#include <curses.h>
#define EXTERN extern
#include "edldef.h"

menudisp()
{
        int i;
        menuwin = newwin(20,80,0,0);
        mvwprintw(menuwin, 1, 33, "FILM/VIDEO MENU");
        mvwprintw(menuwin, 2, 26, "IN");
        mvwprintw(menuwin, 2, 39, "OUT");
        mvwprintw(menuwin, 2, 50, "DURATION");
        mvwprintw(menuwin, 2, 68, "TIME-CODE");
        mvwprintw(menuwin, 4, 15, "R-VTR");
        mvwprintw(menuwin, 4, 62, "/STP");
        for(i=6; i<11; i++)
                mvwprintw(menuwin, i, 62, "/STP");
        for(i=1; i<7; i++)
                mvwprintw(menuwin, i+5, 15, "%c-00%d", (char)(64+i), i);
        mvwprintw(menuwin, 12, 15, "AUX");
        mvwprintw(menuwin, 13, 15, "BLACK");
        mvwprintw(menuwin, 12, 70, "PUNCH ON");
        mvwprintw(menuwin, 9, 2, "EVENT #");
        wmove(menuwin, 19,0);
        for(i=1; i<9; i++)
                wprintw(menuwin, "----------");
        wrefresh(menuwin);
        return(1);
}
```

```
/*  edlcmd.c ---- create cmdwin and prompt user for filename  */
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"

edlcmd()
{
        int c;

        cmdwin = newwin(3, 80, 42, 0);
        wprintw(cmdwin, "Please enter 1) for oldfile or 2) for newfile : ");
        wrefresh(cmdwin);
        c = wgetch(cmdwin);
        while (c!='1' && c!='2') {
                mvwdelch(cmdwin, 0, 48);
                wmove(cmdwin,0,48);
                wrefresh(cmdwin);
                c = wgetch(cmdwin);
                }
        mvwprintw(cmdwin, 1, 0, "ENTER FILENAME : ");
        wrefresh(cmdwin);
        wgetstr(cmdwin,filename);
        if (c=='1') {
                while ((fd = open(filename,0)) == -1){
                        /* test if file exists */
                        werase(cmdwin);
                        mvwprintw(cmdwin,0,0,"%s does not exist.",filename);
                        mvwprintw(cmdwin,1,0,"ENTER FILENAME : ");
                        wrefresh(cmdwin);
                        wgetstr(cmdwin,filename);
                        }
                }
        else    {
                fd = creat(filename,RWMODE);    /* create a new file */
                newfile = TRUE;
                }
        close(fd);
        lnstart = 0;                            /*** initialize lnstart ***/
}
```

```
/*  readedl.c ---- copy EDL file into memory  */
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"

readedl()
{
        int i=0;
        char linebuf[80], *p, *calloc();

        fd = open(filename, 0);
        while (read(fd, linebuf, EDLLEN) != Eof) {
                p = calloc(1,EDLLEN);
                strcpy(p,linebuf);
                edlptr[i] = p;
                i++;
                }
        num_of_ln = i;
        edlptr[i] = NULL;
        if (newfile == TRUE) {maxev = 0;}       /* define max event */
        else sscanf(p, "%d", &maxev);
        close(fd);
}
```

```
/* edldisp.c ---- create edlwin, display EDL from top of file */
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"

edldisp()
{
        edlwin = newwin(20,80,20,0);
        lnstart = 0;
        disp();
}

disp()          /** display 20 lines of edits **/
{
        int i=0;

        while(lnstart+i < lnstart+20) {
                if (edlptr[lnstart+i] != NULL) {
                        mvwprintw(edlwin, i, 0, " %s", edlptr[lnstart+i]);
                        i++;
                        }
                else i=20;
                }
        wclrtobot(edlwin);      /** clear rest of window **/
        lnend = lnstart + 19;   /** save index of last line displayed **/
        wrefresh(edlwin);
}
```

```
/* edledit.c ---- edit mode */
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"
#define CTRL(c) ('c'&037)

char    buf[15];

edledit()
{
        int c, i;

        curry = YRVTR;  /** default y-coordinate of cursor position **/
        tmode = 'C';    /** default transition type **/
        curln = 0;
        if (newfile == TRUE) { }
        else shwevnt();         /** display 1st event on menuwin **/
        keypad(menuwin,TRUE);   /** enable key pad **/
        noecho();
        while(1) {              /** infinite looping **/
                wmove(menuwin, curry, XVTR);
                wrefresh(menuwin);
                c = getch();
                mvwprintw(menuwin, YINPUT, XCOL1, LBLANK);
                wrefresh(menuwin);
                switch(c) {
                case '1':               /*** event selection for display ***/
                        evnt();
                        curry = YRVTR;
                        break;
                case 'a':               /*** VTR selection ***/
                        curry = YRVTR;
                        break;
                case 's':
                        curry = YAVTR;
                        break;
                case 'd':
                        curry = YAVTR+1;
                        break;
                case 'f':
                        curry = YAVTR+2;
                        break;
                case 'g':
                        curry = YAVTR+3;
                        break;
                case 'h':
                        curry = YAVTR+4;
                        break;
                case 'j':
                        curry = YAVTR+5;
                        break;
                case 'k':
                        curry = YAVTR+6;
                        break;
                case 'l':
                        curry = YAVTR+7;
                        break;
                case 'y':               /*** audio/video mode ***/
                        mvwprintw(menuwin, YAVMODE, XAVMODE, "A/V");
                        wrefresh(menuwin);
                        strcpy(avmo,"B");
                        curry = YRVTR;
                        break;
                case 'i':               /*** video-only ***/
                        mvwprintw(menuwin, YAVMODE, XAVMODE, "V  ");
```

```
            wrefresh(menuwin);
            strcpy(avmo,"V");
            curry = YRVTR;
            break;
        case 'u':               /** audio-only, prompt for track # **/
            audin();
            curry = YRVTR;
            break;
        case 'p':               /*** CUT ***/
            mvwprintw(menuwin, YTRNMD, XCOL1, "CUT     ");
            wrefresh(menuwin);
            tmode = 'C';
            curry = YRVTR;
            break;
        case '[':               /*** DISSOLVE ***/
            mvwprintw(menuwin, YTRNMD, XCOL1, "FADE    ");
            wrefresh(menuwin);
            tmode = 'D';
            curry = YRVTR;
            break;
        case ']':               /*** WIPE ***/
            mvwprintw(menuwin, YTRNMD, XCOL1, "WIPE-   ");
            wrefresh(menuwin);
            tmode = 'W';
            getwcode();     /*** get wipe code ***/
            curry = YRVTR;
            break;
        case '\\':              /*** KEY ***/
            mvwprintw(menuwin, YTRNMD, XCOL1, "KEY     ");
            wrefresh(menuwin);
            tmode = 'K';
            curry = YRVTR;
            break;
        case '8':                       /** OPEN END **/
            for (i=0; i<10; i++) {
                    mvwprintw(menuwin, YRVTR+i, XOUT, SHBLANK);
                    mvwprintw(menuwin, YRVTR+i, XDUR, SHBLANK);
                    }
            wrefresh(menuwin);
            curry = YRVTR;
            break;
        case 'x':                       /** SET IN **/
            currx = XIN;
            settc(ptci);
            break;
        case 'c':                       /** SET OUT **/
            currx = XOUT;
            settc(ptco);
            break;
        case 'z':                       /** ENTER EDIT DURATION **/
            mvwprintw(menuwin,YINPUT,XCOL1,"DURATION : ");
            wrefresh(menuwin);
            sign = '+';
            strcpy(tcdur,"00:00:");
            if (getdur(tcdur)==0) break;
            readscr(menuwin,curry,XIN,TCLEN,ptci);
            tcadd(ptci,tcdur);
            tcprin(curry,XOUT);
            mvwprintw(menuwin,curry,XDUR,"%s",tcdur);
            wrefresh(menuwin);
            break;
        case 'v':                       /** TRIM IN **/
            mvwprintw(menuwin,YINPUT,XCOL1,"TRIM IN  : ");
            wrefresh(menuwin);
            trimio(ptci,XIN);
            break;
```

```
                case 'b':                       /** TRIM OUT **/
                        mvwprintw(menuwin,YINPUT,XCOL1,"TRIM OUT : ");
                        wrefresh(menuwin);
                        trimio(ptco,XOUT);
                        break;
                case CTRL(L):                   /** save edl input **/
                        if (getsrc()==1) {
                                if (save()==1) {
                                        upedlwin();
                                        upmenu();
                                        }
                                else {
                                        upedlwin();
                                        mvwprintw(menuwin,YINPUT,XCOL1,
                                "ERROR--ILLEGAL INPUTS--FUNCTION FAILED");
                                        wrefresh(menuwin);
                                        }
                                }
                        curry = YRVTR;
                        break;
                case '6':                               /** save on disk **/
                        todisk();
                        mvwprintw(menuwin,YINPUT,XCOL1,"DATA SAVED ON DISK");
                        wrefresh(menuwin);
                        curry = YRVTR;
                        break;
                case '.':                               /** delete event **/
                        if (ridevnt()==1)
                                upedlwin();
                        curry = YRVTR;
                        break;
                case 'Q':                               /*** QUIT ***/
                        mvwprintw(menuwin,YINPUT,XCOL1,"SAVE DATA? (y/n): ");
                        wrefresh(menuwin);
                        if (request()==1) todisk();
                        mvwprintw(menuwin,YINPUT,XCOL1,"QUIT ?    (y/n) : ");
                        wrefresh(menuwin);
                        if (request()==1) return;
                        curry = YRVTR;
                        break;
                }
        }
}
```

```
/***  evnt.c  ***/
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"

char buf[5];

/*** prompt for event # and display ***/
evnt()
{
        int m=0;
        char *fmt;

        strcpy(buf,"");
        echo();
        mvwprintw(menuwin, YINPUT, XCOL1, "EVENT # : ");
        while (m!=1 || evnum<1) {
                mvwprintw(menuwin, YINPUT, XCOL1+10, SHBLANK);
                wrefresh(menuwin);
                mvwgetstr(menuwin, YINPUT, XCOL1+10, buf);
                if (strcmp(buf,"")==0) {noecho(); return;}
                m = sscanf(buf, "%d", &evnum);
                }

        if (evnum>MAXEVNUM) {           /*** test for validity of event # ***/
                mvwprintw(menuwin, YINPUT, XCOL1, "EVENT # %d IS TOO BIG", evnum);
                wrefresh(menuwin);
                noecho();
                return;
                }

        if (findev()==0) {              /*** event does not exist ***/
                curev = evnum;
                if (curev>99) fmt = "%d";
                if (curev>9 && curev<100) fmt = "0%d";
                if (curev<10) fmt = "00%d";
                sprintf(evstr,fmt,curev);
                clrmenu();              /** clear old data for new inputs **/
                mvwprintw(menuwin, YEVNUM, XEVNUM, fmt, curev);
                wrefresh(menuwin);
                }
        else    {                               /** event # exists **/
                clrmenu();                      /** clear menu **/
                shwevnt();                      /** show event **/
                upedlwin();                     /** update edlwin **/
                }
        noecho();
}


/*** event lookup : 1) if event does not exist, return 0.
                       if event exists, return 1.
                    2) save index of location where event should belong. ***/
findev()
{
        int i=0, tmpevnt;

        while (edlptr[i] != NULL) {
                sscanf(edlptr[i],"%d",&tmpevnt);
                if (tmpevnt < evnum) {
                        i++;
                        continue;
                        }
                if (tmpevnt == evnum) {
                        curln = i;
```

```
                        return(1);
                        }
                if (tmpevnt > evnum) {
                        curln = i;
                        return(0);
                        }
                }
        curln = i;
        return(0);
}

/*** clear menu, show default values. ***/
clrmenu()
{
        int i;

        mvwprintw(menuwin, YAVMODE, XAVMODE, "A/V");
        mvwprintw(menuwin, YTRNMD, XCOL1, "CUT     ");
        tmode = 'C';
        mvwprintw(menuwin, YSRCDST, XCOL1, SHBLANK);
        mvwprintw(menuwin, YTRNLEN, XCOL1, SHBLANK);
        for (i=0; i<10; i++)
                mvwprintw(menuwin, YRVTR+i, XIN, LBLANK);
        wrefresh(menuwin);
}


/*** update edlwin to show EDLs of interest ***/
upedlwin()
{
        if (curln >= lnend) {
                lnstart = curln - 9;
                }
        else if (curln < lnstart) {
                lnstart = curln-1;
                }
        disp();
}
```

```
/* shwevnt.c */
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"


/*** show data of selected event ***/
shwevnt()
{
        int cury;
        char tmp;

        shwedl();
        strcpy(ortci,rtci);
        if (edlptr[curln+1]==NULL) {return;}
        else    sscanf(edlptr[curln+1], "%d", &evnum);
        if (evnum==curev) {             /** show 2nd line of a 2-line edit **/
                oreel = reel;
                curln++;
                shwedl();
                mvwprintw(menuwin, YRVTR, XIN, "%s", ortci);
                wrefresh(menuwin);
                tcsub(rtco,ortci);
                cury = YRVTR;
                tcprin(cury,XDUR);
                if (oreel<7) {tmp = 'A'+oreel-1; sprintf(src, "%c", tmp);}
                if (reel<7) {tmp = 'A'+reel-1; sprintf(dst, "%c", tmp);}
                if (oreel==7) strcpy(src,"AX");
                if (reel==7) strcpy(dst,"AX");
                if (oreel==8) strcpy(src,"BL");
                if (reel==8) strcpy(dst,"BL");
                mvwprintw(menuwin, YSRCDST, XCOL1, "%s TO %s", src, dst);
                wrefresh(menuwin);
                }
}


/*** show one line of edit ***/
shwedl()
{
        int cury;
        char *fmt;

        sscanf(edlptr[curln],"%3s %3s %2s %4s %5s %11s %11s %11s %11s",
                evstr, rlstr ,avmo,tmdstr,tdur,ptci,ptco,rtci,rtco);

        sscanf(evstr,"%d",&curev);

        if (sscanf(tmdstr,"%c%d",&tmode,&wcode) != 2)
                sscanf(tmdstr,"%c",&tmode);

        if (strcmp(rlstr,"AX") == 0) reel = 7;
        if (strcmp(rlstr,"BL") == 0) {reel = 8;}
        else sscanf(rlstr, "%d", &reel);

        mvwprintw(menuwin, YEVNUM, XEVNUM, "%s" , evstr);
        wrefresh(menuwin);

        if (strcmp(avmo,"B") == 0) {
                mvwprintw(menuwin, YAVMODE, XAVMODE, "A/V");
                }
        else mvwprintw(menuwin, YAVMODE, XAVMODE, "%s  ", avmo);
        wrefresh(menuwin);

        switch(tmode) {
```

```
        case 'C':
                fmt="CUT";
                break;
        case 'W':
                fmt="WIPE-";
                break;
        case 'K':
                fmt="KEY";
                break;
        case 'D':
                fmt="FADE";
                break;
        }
        mvwprintw(menuwin, YTRNMD, XCOL1, fmt);
        if (tmode=='W') {
                if (wcode<10) fmt="00%d";
                if (wcode>99) {fmt="%d";}
                else fmt="0%d";
                wprintw(menuwin,fmt,wcode);
                }
        wrefresh(menuwin);

        if (strcmp(tdur,"00:00") != 0) {
                mvwprintw(menuwin, YTRNLEN, XCOL1, "%s", tdur);
                wrefresh(menuwin);
                }

        cury = reel==1 ? YAVTR : YAVTR + reel-1;
        mvwprintw(menuwin, cury, XIN, "%s", ptci);
        mvwprintw(menuwin, cury, XOUT, "%s", ptco);
        wrefresh(menuwin);
        tcsub(ptco,ptci);
        tcprin(cury,XDUR);

        mvwprintw(menuwin, YRVTR, XIN, "%s", rtci);
        mvwprintw(menuwin, YRVTR, XOUT, "%s", rtco);
        wrefresh(menuwin);
        tcsub(rtco,rtci);
        cury = YRVTR;
        tcprin(cury,XDUR);

}


/*** subtract tc2 from tc1 ***/
tcsub(tc1, tc2)
        char *tc1,*tc2;
{
        int hr1, min1, sec1, fr1;
        int hr2, min2, sec2, fr2;
        int dhr, dmin, dsec, dfr;

        sscanf(tc1, "%d:%d:%d:%d", &hr1,&min1,&sec1,&fr1);
        sscanf(tc2, "%d:%d:%d:%d", &hr2,&min2,&sec2,&fr2);

        if (fr1<fr2) {
                sec1--;
                fr1 += 30;
                }
        dfr = fr1 - fr2;
        if (sec1<sec2) {
                min1--;
                sec1 += 60;
                }
        dsec = sec1 - sec2;
        if (min1<min2) {
```

```
                hr1--;
                min1 += 60;
                }
        dmin = min1 - min2;
        dhr = hr1 - hr2;

        tmptc[0]=dhr;
        tmptc[1]=dmir.;
        tmptc[2]=dsec;
        tmptc[3]=dfr;
        if (tmptc[0]<0) {return(0);}
        else return(1);
}


/*** print a tc data array in the right format ***/
tcprin(y,x)
        int y,x;
{
        int i;
        char *fmt;

        fmt = tmptc[0]<10 ? "0%d" : "%d";
        mvwprintw(menuwin,y,x,fmt,tmptc[0]);
        wrefresh(menuwin);
        for (i=1; i<4; i++) {
                fmt = tmptc[i]<10 ? ":0%d" : ":%d";
                wprintw(menuwin,fmt,tmptc[i]);
                wrefresh(menuwin);
                }
}
```

```
/***  rout.c  ***/
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"
char buf[15];


/*** prompt for audio track specification ***/
audin()
{
        echo();
        strcpy(buf,"");
        mvwprintw(menuwin, YINPUT, XCOL1, "A1 OR A2 : ");
        while (strcmp(buf,"A1")!=0 && strcmp(buf,"A2")!=0) {
                mvwprintw(menuwin, YINPUT, XCOL1+11, SHBLANK);
                wrefresh(menuwin);
                mvwgetstr(menuwin, YINPUT, XCOL1+11, buf);
                if (strcmp(buf,"")==0) {noecho(); return;}
                }
        mvwprintw(menuwin, YAVMODE, XAVMODE, "%2s ", buf);
        wrefresh(menuwin);
        noecho();
}


/*** prompt for wipe code ***/
getwcode()
{
        int m=0;
        char *fmt;

        echo();
        mvwprintw(menuwin, YINPUT, XCOL1, "WIPE CODE : ");
        while (m!=1) {
                mvwprintw(menuwin, YINPUT, XCOL1+12, SHBLANK);
                wmove(menuwin, YINPUT, XCOL1+12);
                wrefresh(menuwin);
                wgetstr(menuwin,buf);
                m = sscanf(buf,"%d",&wcode);
                }
        if (wcode<10) {fmt = "00%d";}
        else    fmt = wcode>99 ? "%d" : "0%d";
        mvwprintw(menuwin, YTRNMD, XCOL1+5, fmt, wcode);
        wrefresh(menuwin);
        noecho();
}


/*** call tcinput, set tc and calculates duration ***/
settc(tc)
        char *tc;
{                                               /** tc is either ptci or ptco **/
        readscr(menuwin, curry, XIN, TCLEN, ptci);
        readscr(menuwin, curry, XOUT, TCLEN, ptco);
        if (tcinput(tc,curry,currx)==0) return;
        if (strcmp(ptci,MTTC)==0 || strcmp(ptco,MTTC)==0)
        /** either or both tc's are missing, cannot calculate duration **/
                return;
        if (tcsub(ptco,ptci)==0) {      /** time in > time out, blank duration **/
                mvwprintw(menuwin,curry,XDUR,MTTC);
                wrefresh(menuwin);
                }
        else tcprin(curry,XDUR);
}
```

```
/*** prompt for time code input with or without colons ***/
tcinput(tc,y,x)
        char *tc;
        int y, x;
{
        int m=0;

        strcpy(buf,"");
        echo();
        if (x==XIN) mvwprintw(menuwin,YINPUT,XCOL1,"ENTER IN POINT : ");
        if (x==XOUT) mvwprintw(menuwin,YINPUT,XCOL1,"ENTER OUT POINT: ");
        while(m!=4) {
                mvwprintw(menuwin, YINPUT, XCOL1+17, SHBLANK);
                wmove(menuwin,YINPUT,XCOL1+17);
                wrefresh(menuwin);
                wgetstr(menuwin,buf);
                if (strcmp(buf,"")==0) {noecho(); return(0);}
                if (strlen(buf)==8) {
                        m = sscanf(buf,"%2d%2d%2d%2d",
                                &tmptc[0],&tmptc[1],&tmptc[2],&tmptc[3]);
                        }
                else m = sscanf(buf,"%d:%d:%d:%d",
                                &tmptc[0],&tmptc[1],&tmptc[2],&tmptc[3]);
                }
        tcprin(YRVTR+1,XIN);
        readscr(menuwin,YRVTR+1,XIN,TCLEN,tc);
        mvwprintw(menuwin,YRVTR+1,XIN,MTTC);
        mvwprintw(menuwin,y,x,"%s",tc);
        wrefresh(menuwin);
        noecho();
        return(1);
}

getdur(dur)                     /** get edit duration or tc offset **/
        char *dur;
{
        int sec=0, fr=0, m=0;
        char *fmt;

        strcpy(buf,"");
        echo();
        while (m!=1 && m!=2) {
                mvwprintw(menuwin, YINPUT, XCOL1+11, SHBLANK);
                wmove(menuwin, YINPUT, XCOL1+11);
                wrefresh(menuwin);
                wgetstr(menuwin,buf);
                if (strcmp(buf,"")==0) {noecho(); return(0);}
                if ((m=sscanf(buf,"%d:%d",&sec,&fr)) != 2) {
                        m = sscanf(buf,"%d",&fr);
                        sec = 0;
                        }
                if (sign=='+') {
                        if (sec<0 || fr<0) m=0;
                        }
                }

        if (sign=='\0') {
                if (sec<0 || fr<0) {
                        sign = '-';
                        if (sec<0) sec = -sec;
                        if (fr<0) fr = -fr;
                        }
                else sign = '+';
                }
```

```
        if (m==1 && fr>29) {sec=fr/30; fr=fr%30;}
        if (sec<10) strcat(dur,"0");
        fmt = fr<10 ? "%d:0%d" : "%d:%d" ;
        sprintf(buf,fmt,sec,fr);
        strcat(dur,buf);
        noecho();
        return(1);
}

tcadd(tc,dtc)                   /** increment tc by duration or offset dtc **/
        char *tc, *dtc;         /**     and put it into tmptc     **/
{
        int hr, min, sec, fr;
        int dhr, dmin, dsec, dfr;
        int shr, smin, ssec, sfr;

        sscanf(tc,"%d:%d:%d:%d",&hr,&min,&sec,&fr);
        sscanf(dtc,"%d:%d:%d:%d",&dhr,&dmin,&dsec,&dfr);

        shr = hr + dhr;
        smin = min + dmin;
        ssec = sec + dsec;
        sfr = fr + dfr;

        if (sfr>29) {ssec++; sfr -= 30;}
        if (ssec>59) {smin++; ssec -= 60;}
        if (smin>59) {shr++; smin -= 60;}

        tmptc[0] = shr;
        tmptc[1] = smin;
        tmptc[2] = ssec;
        tmptc[3] = sfr;
}

trimio(tc,x)                    /** trim in and out **/
        char *tc;
        int x;
{
        sign = '\0';
        strcpy(tcdur,"00:00:");
        if (getdur(tcdur)==0) return;
        readscr(menuwin,curry,x,TCLEN,tc);
        if (sign=='+') tcadd(tc,tcdur);
        if (sign=='-') {
                if (tcsub(tc,tcdur)==0) return;
                }
        tcprin(curry,x);
        readscr(menuwin,curry,XIN,TCLEN,ptci);
        readscr(menuwin,curry,XOUT,TCLEN,ptco);
        if (strcmp(ptci,MTTC)==0 || strcmp(ptco,MTTC)==0)
                return;
        if (tcsub(ptco,ptci)==0) {
                mvwprintw(menuwin,curry,XDUR,MTTC);
                wrefresh(menuwin);
                }
        else tcprin(curry,XDUR);
}
```

```
/*
readscr.c - get a string from window structure
*/

#include <curses.h>

int
readscr( win, y, x, len, str )

WINDOW *win;        /* window to read from */
int x, y;           /* starting window location of string */
int len;            /* size of string */
char *str;          /* where to put NULL-terminated result */
{
        char *tbuf;
        char c;
        char *malloc();
        int index;
        int tx,ty;

        if( win==NULL ) return(-1);
        if( len<0 ) return(-1);

        tbuf = (char *)malloc( len+1 );
        if( tbuf==NULL ) return(-1);

        getyx( win, ty, tx );

        for( index=0; index<len; index++ ) {
                c = (char)( mvwinch(win,y,x++) & A_CHARTEXT );
                tbuf[index]=c;
                }

        tbuf[len]='\0';

        strcpy( str, tbuf );
        free( (char *) tbuf );

        wmove( win, ty, tx );

        return(0);
}
```

```
/** getdat.c  **/
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"
#define CAT2SPACES  strcat(linebuf,"  ")


/*** read data from menu to compose one line of edit ***/
getdat1(linebuf)
        char *linebuf;
{
        char c;

        strcpy(linebuf,evstr);
        CAT2SPACES;
        if (strcmp(src,"AX")==0) {strcat(linebuf,"AX "); curry = YAVTR + 6;}
        if (strcmp(src,"BL")==0) {strcat(linebuf,"BL "); curry = YAVTR + 7;}
        if (strlen(src)==1) {
                sscanf(src,"%c",&c);
                curry = YAVTR + (int)(c-'A');
                readscr(menuwin,curry,XVTR+2,3,rlstr);
                strcat(linebuf,rlstr);
                }
        ysrc = curry;
        ydst = 0;
        CAT2SPACES;
        readscr(menuwin,YAVMODE,XAVMODE,3,avmo);
        if (strcmp(avmo,"A/V")==0) strcpy(avmo,"B  ");
        if (strcmp(avmo,"V")==0) strcpy(avmo,"V  ");
        strcat(linebuf,avmo);
        CAT2SPACES;
        strcat(linebuf,"C     00:00  ");
        readscr(menuwin,curry,XIN,TCLEN,ptci);
        strcat(linebuf,ptci);
        CAT2SPACES;
        readscr(menuwin,curry,XOUT,TCLEN,ptco);
        strcat(linebuf,ptco);
        CAT2SPACES;
        readscr(menuwin,YRVTR,XIN,TCLEN,rtci);
        strcat(linebuf,rtci);
        CAT2SPACES;
        readscr(menuwin,curry,XDUR,TCLEN,tcdur);
        tcadd(rtci,tcdur);
        if (tmptc[1]>59 || tmptc[2]>59 || tmptc[3]>29) return(-1);
        if (tmode=='C') {
                tcprin(YRVTR,XOUT);
                wrefresh(menuwin);
                readscr(menuwin,YRVTR,XOUT,TCLEN,rtco);
                }
                else    {
                        tcprin(YRVTR+1,XOUT);
                        readscr(menuwin,YRVTR+1,XOUT,TCLEN,rtco);
                        mvwprintw(menuwin,YRVTR+1,XOUT,MTTC);
                        }
        strcat(linebuf,rtco);
        strcat(linebuf,"\n");
        return(1);
}


/*** compose 2nd line of edit from data on menuwin ***/
getdat2(linebuf)
        char *linebuf;
{
        char c;
```

```
        strcpy(linebuf,evstr);
        CAT2SPACES;
        if (strcmp(dst,"AX")==0) {strcat(linebuf,"AX "); curry = YAVTR + 6;}
        if (strcmp(dst,"BL")==0) {strcat(linebuf,"BL "); curry = YAVTR + 7;}
        if (strlen(dst)==1) {
                sscanf(dst,"%c",&c);
                curry = YAVTR + (int)(c-'A');
                readscr(menuwin,curry,XVTR+2,3,rlstr);
                strcat(linebuf,rlstr);
                }
        ydst = curry;
        CAT2SPACES;
        strcat(linebuf,avmo);
        CAT2SPACES;
        if (tmode!='W') {sprintf(tmdstr,"%c   ",tmode); strcat(linebuf,tmdstr);}
        else    {
                readscr(menuwin,YTRNMD,XCOL1+5,3,tmdstr);
                strcat(linebuf,"W");
                strcat(linebuf,tmdstr);
                }
        CAT2SPACES;
        strcat(linebuf,tdur);
        CAT2SPACES;
        readscr(menuwin,curry,XIN,TCLEN,ptci);
        strcat(linebuf,ptci);
        CAT2SPACES;
        readscr(menuwin,curry,XOUT,TCLEN,ptco);
        strcat(linebuf,ptco);
        CAT2SPACES;
        strcpy(rtci,rtco);
        strcat(linebuf,rtci);
        CAT2SPACES;
        readscr(menuwin,curry,XDUR,TCLEN,tcdur);
        tcadd(rtci,tcdur);
        if (tmptc[1]>59 || tmptc[2]>59 || tmptc[3]>29) return(-1);
        tcprin(YRVTR,XOUT);
        wrefresh(menuwin);
        readscr(menuwin,YRVTR,XOUT,TCLEN,rtco);
        strcat(linebuf,rtco);
        strcat(linebuf,"\n");
        return(1);
}


/*** write EDLs from buffer to file ***/
todisk()
{
        int i=0;
        fd = open(filename,1);
        while (edlptr[i]!=NULL) {
                write(fd,edlptr[i],EDLLEN);
                i++;
                }
        close(fd);
}


/*** prompt for a boolean input ***/
request()
{
        char c = NULL;

        while (c!='y' && c!='n') {
                mvwprintw(menuwin,YINPUT,XCOL1+18,SHBLANK);
                wmove(menuwin,YINPUT,XCOL1+18);
```

```
                wrefresh(menuwin);
                c = wgetch(menuwin);
                }
        if (c=='y') return(1);
        return(0);
}
```

```
/** save.c takes care of routines called by CTRL(L) **/
#include <curses.h>
#define EXTERN  extern
#include "edldef.h"
#include "locdef.h"

char buf[5];
char edlbuf[80], *p, *calloc();

/*** call getvtr ***/
getsrc()
{
        mvwprintw(menuwin,YINPUT,XCOL1,"ENTER SRC : ");         /** source 1 **/
        wrefresh(menuwin);
        if (getvtr(src)==0) return(0);
        if (tmode != 'C') {
                                /** source 2 for a 2-line edit **/
                mvwprintw(menuwin,YINPUT,XCOL1,"ENTER DST : ");
                wrefresh(menuwin);
                if (getvtr(dst)==0) return(0);
                if (strcmp(src,dst)==0) {
                        mvwprintw(menuwin,YINPUT,XCOL1,"SRC = DST : ILLEGAL INPUTS");
                        wrefresh(menuwin);
                        return(-1);
                        }
                mvwprintw(menuwin,YINPUT,XCOL1,"TRANSITION:");
                wrefresh(menuwin);      /** prompt for transition duration **/
                strcpy(tdur,"");                /** for cases other than a CUT **/
                if (getdur(tdur)==0) return(0);
                }
        mvwprintw(menuwin,YSRCDST,XCOL1,SHBLANK);
        mvwprintw(menuwin,YTRNLEN,XCOL1,SHBLANK);
        if (tmode!='C') {
                mvwprintw(menuwin,YSRCDST,XCOL1,"%s TO %s",src,dst);
                mvwprintw(menuwin,YTRNLEN,XCOL1,"%s",tdur);
                }
        else mvwprintw(menuwin,YSRCDST,XCOL1,"%s",src);
        wrefresh(menuwin);
        return(1);
}


/*** prompt for VTR designation ***/
getvtr(vtr)
        char *vtr;
{
        char c;

        strcpy(buf,"");
        echo();
        while(1) {
                mvwprintw(menuwin,YINPUT,XCOL1+12,SHBLANK);
                wmove(menuwin,YINPUT,XCOL1+12);
                wrefresh(menuwin);
                wgetstr(menuwin,buf);
                if (strcmp(buf,"")==0) {noecho();return(0);}
                if (strlen(buf)==2) {
                        if (strcmp(buf,"AX")==0) {
                                strcpy(vtr,"AX");
                                noecho();
                                return(1);
                                }
                        if (strcmp(buf,"BL")==0) {
                                strcpy(vtr,"BL");
                                noecho();
                                return(1);
```

```
                        }
                continue;
                }
        if (strlen(buf)==1) {
                sscanf(buf,"%c",&c);
                if (c>='A' && c<='F') {
                        strcpy(vtr,buf);
                        noecho();
                        return(1);
                        }
                continue;
                }
        }
}


/*** save event input on menu into buffer ***/
save()
{
        if (getdat1(edlbuf)==-1) return(-1);
        p = calloc(1,EDLLEN);
        strcpy(p,edlbuf);
        evnum = curev;
        if (findev()==0) {      /** data are a new addition to the list **/
                insrtev(curln);
                num_of_ln++;
                if (maxev<curev) maxev=curev;
                if (tmode!='C') {
                        if (getdat2(edlbuf)==-1) return(-1);
                        p = calloc(1,EDLLEN);
                        strcpy(p,edlbuf);
                        curln++;
                        insrtev(curln);
                        num_of_ln++;
                        }
                return(1);
                }
        else {                          /** findev()==1; replacing old data **/
                edlptr[curln] = p;
                if (edlptr[curln+1]==NULL) {evnum=0;}
                else    sscanf(edlptr[curln+1],"%d",&evnum);
                if (tmode=='C') {
                        if (evnum==curev) {delev(curln+1); num_of_ln--;}
                        /** 2 -> 1 replacement **/
                        return(1);
                        /** 1 -> 1 replacement **/
                        }
                else {
                        if (getdat2(edlbuf)==-1) return(-1);
                        p = calloc(1,EDLLEN);
                        strcpy(p,edlbuf);
                        curln++;
                        if (evnum==curev) {edlptr[curln] = p;}
                        /** 2 -> 2 replacement **/
                        else {insrtev(curln); num_of_ln++;}
                                /** 1 -> 2 replacement **/
                        return(1);
                }
        }
}


/*** delete event of index ln ***/
delev(ln)
        int ln;
{
```

```
        int i=ln;
        while(i<num_of_ln) {
                edlptr[i]=edlptr[i+1];
                i++;
                }
        edlptr[i]=NULL;
}


/*** insert event at index ln ***/
insrtev(ln)
        int ln;
{
        int i=num_of_ln;
        while(i>=ln) {
                edlptr[i+1]=edlptr[i];
                i--;
                }
        edlptr[ln]=p;
}


/*** update menu for next event input ***/
upmenu()
{
        int i;
        char *fmt;
        if (maxev==MAXEVNUM) {
                /* if the last edit is event 512, show it on menu. */
                evnum=curev;
                findev();
                clrmenu();
                shwevnt();
                return;
                }
        readscr(menuwin,ysrc,XOUT,TCLEN,ptci);
        readscr(menuwin,ydst,XOUT,TCLEN,ptco);
        for (i=0; i<10; i++)
                mvwprintw(menuwin,YRVTR+i,XIN,LBLANK);
        mvwprintw(menuwin,YRVTR,XIN,"%s",rtco);
                        /** assign time out to time in **/
        mvwprintw(menuwin,YRVTR,XOUT,"%s",rtco);        /** blank duration **/
        mvwprintw(menuwin,ysrc,XIN,"%s",ptci);
        mvwprintw(menuwin,ysrc,XOUT,"%s",ptci);
        if (ydst!=0) {
                mvwprintw(menuwin,ydst,XIN,"%s",ptco);
                mvwprintw(menuwin,ydst,XOUT,"%s",ptco);
                }
        wrefresh(menuwin);
        curev = maxev + 1;
        if (curev>9 && curev<100) {fmt="0%d";}
        else fmt = curev<10 ? "00%d" : "%d" ;
        sprintf(evstr,fmt,curev);
        mvwprintw(menuwin,YEVNUM,XEVNUM,fmt,curev);
        wrefresh(menuwin);
}


/*** prompt for an event # to be deleted ***/
ridevnt()
{
        int m=0;

        strcpy(buf,"");
        echo();
        mvwprintw(menuwin,YINPUT,XCOL1,"DELETE EVENT # : ");
```

```
        while (m!=1) {
                mvwprintw(menuwin,YINPUT,XCOL1+17,SHBLANK);
                wrefresh(menuwin);
                mvwgetstr(menuwin,YINPUT,XCOL1+17,buf);
                if (strcmp(buf,"")==0) {noecho(); return(0);}
                m = sscanf(buf,"%d",&evnum);
                if (m==1 && (evnum<1 || evnum>maxev)) m=0;
                }
        noecho();
        if (findev()==0) {
                mvwprintw(menuwin,YINPUT,XCOL1,"EVENT #%d DOES NOT EXIST", evnum);
                wrefresh(menuwin);
                return(0);
                }
        delev(curln);
        num_of_ln--;
        if (findev()==1) {
                delev(curln);
                num_of_ln--;
                }
        return(1);
}
```

REFERENCE

1. Video Editing And Post-Production
   Gary H. Anderson
   NY: Knowledge Industry Publications, Inc.,1984.

2. The C Programming Language
   Kernighan / Ritchie
   NJ: Prentice-Hall, Inc.,1978.

3. Recording Studio Handbook :
   Time Code Implementation
   John M. Woram

4. CMX User Guide

5. Television Broadcasting :
   Video and Audio Signal Distribution

6. Videodisc Editing System
   Russ Sasnett, Peter Roos
   (MIT Film/Video)

7. The Time Code Book
   EECO incorporated

8. The Impact of Optical Videodiscs on Filmmaking
   Nicholas Negropoate
   (MIT Professor Of Computer Graphics)

9. Horizontal And Vertical Blanking,
   The 4 Color Fields and Subcarrier Phase
   ABC Tech Notes Feb. 79

10. Editing By The 'Scope
    Diana Weynand
    EITV / OCTOBER / 1984

11. VCRs : Very Cloudy Regions
    Carl Bentz
    Video Systems April 1985

12. The Fundamentals of Television
    Rex H. Stevens International Television

13. PC-TV
    William Claxton
    PC World February 1984

14. SMPTE Journal, July 1982

15. 'SMPTE' The Fusion Force For Audio And Video
    Frank Serafine
    R-e/p 108 April 1981

16. Basic System Timing
Sally Wells
International Television May 1985

17. Videodiscs And Optical Storage
Andy Lippman
(MIT Achitecture Machine Group)