

**R.V. COLLEGE OF ENGINEERING
BANGALORE**

**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING**

SYSTEM SOFTWARE LAB

PROJECT REPORT

**A FULL SCREEN TEXT EDITOR
UNDER THE LINUX OPERATING
SYSTEM**

**DEVELOPED BY
SRIVAS N. CHENNU
1RV98CS086
5TH SEMESTER CSE
R.V.C.E.**

**R.V. COLLEGE OF ENGINEERING
BANGALORE**

**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING**

CERTIFICATE

This is to certify that this project work on a screen editor has been successfully completed by

SRIVAS N. CHENNU

On a partial fulfillment of 5th sem BE, Computer science, System Software Lab during the year 2000-2001 in the department as prescribed by VTU

STAFF INCHARGE

Dept. of CSE

PROF. K. A. RANGANATHA SETTY

H.O.D., Dept. of CSE

TABLE OF CONTENTS

1.Introduction.

..... 1.1 Purpose

..... 1.2 Scope

..... 1.3 Definitions, acronyms & abbreviations

..... 1.4 References

..... 1.5 Overview

2. General Description

..... 2.1 Product Perspective

.....2.2 Product Functions

.....2.3 User Characteristics

..... 2.4 General Constraints

..... 2.5 Assumptions & dependencies

3. Functional Requirements

..... Functional Requirements - I

..... Introduction

..... Inputs

..... Processing

..... Outputs

..... Functional Requirements – II

TABLE OF CONTENTS - CONTINUED

4. External Interface Requirements

.....User Interface

.....Hardware Interface

.....Software Interface

5. Performance Requirements

6. Design Constraints

.....Standard Compliance

.....Hardware Limitations

Appendix

.....Source Code Listing

1. INTRODUCTION

STEP – short for **SRIVAS' TEXT EDITING PROGRAM**, is a full screen text editor designed for the **LINUX** operating system. It features an interactive and user-friendly interface, and is equipped with many of the commonly used text editing features.

1.1 PURPOSE

The main purposes this editor intends to fulfill are as follows:

- ✓ Provide users a convenient way to create, open, modify and save their text files.
- ✓ Provide an easy-to-learn and use text-editing software for new **LINUX** users.
- ✓ Provide programmers with a platform for creating and modifying source files of various programming languages.

1.2 SCOPE

The scope of this software is limited to the manipulation of text documents that contain pure ASCII text. Due to this reason, it cannot work with documents created by word-processing software.

Furthermore, **STEP** is designed to run only under the **LINUX** operating system environment. Its design is closely related to the **LINUX** OS programming environment, and hence cannot execute properly in its absence. Its functioning under other **UNIX** clone operating systems is currently untested.

1.3 REFERENCES

The following references were consulted during the preparation of this software

- ❑ For documentation about the C++ Standard Template Library –
C++ Primer – By Stanley B. Lippman and Josie Lajoie
- ❑ The Linux Programmers' Manual and the manual pages in LINUX provided the reference and documentation for programming with the `ncurses` library of functions.
- ❑ For programming reference regarding the C programming language –
The C Programming Language – By Brian W. Kernighan and Dennis M. Ritchie

1.4 OVERVIEW

An overview of the functionality provided by STEP is briefly mentioned below.

1. Users can open files stored on disk, located in any directory, provided that they know the name of the file. Users can also create new files using STEP.
2. Once opened, files can be modified as desired, using the text editing features of STEP. Text block operations, like Cut, Copy, and Paste can also be performed.
3. STEP also allows users to search for information in the opened files. Additionally, a basic level of text formatting can be accomplished using STEP.
4. Once any desired modifications are complete, users can save the modified file back to disk. The edited files can also be saved with a different location and/or name. STEP also allows users to delete files on disk from within the editor.

2. GENERAL DESCRIPTION

A general, detailed description of various features of STEP follows. This section elaborates on the functionality, constraints, dependencies, and provides a comparative estimate of its performance.

2.1 PRODUCT PERSPECTIVE

STEP's performance was evaluated against that of the widely popular vim editor available in Linux. (Using which the STEP was coded). The performance of STEP was found to be equivalent that of vim for files lesser than 10MB in size.

The following performance ratio charts are indicative of the comparative performance of the two editors while opening files of large sizes.

FILE SIZE IN MEGABYTES	NUMBER OF LINES IN FILE	TIME TAKEN BY VIM TO OPEN FILE (SECONDS)	TIME TAKEN BY STEP TO OPEN FILE (SECONDS)
10	653,000	1.5	3
25	1,632,500	7	9
40	2,612,000	22	35

Performance comparison of STEP and VIM

2.2 PRODUCT FUNCTIONS

The following listing explains all the functions and features that STEP provides.

□ **File operations**

These operations involve manipulation of files on the disk.

- Create a new file.
- Open an existing file from the disk.
- Save the current file to disk.
- Save the file current file with a different name and/or path.
- Delete a file from disk.
- Insert a file from the disk into the current buffer.
- Switch between currently open files.

□ **Cursor Movement Operations**

These operations involve moving the cursor to a desired location on the screen

- Move up/down by a line in the current buffer.
- Move up/down by a page in the current buffer.
- Move left/right by a character in the current line.
- Move to the beginning/end of the current line.
- Move to the beginning/end of the current file.

Continued

□ Text Editing Operations

These operations involve modification of the text buffer.

- Insert a character at the current cursor position.
- Delete a character at the current cursor position.
- Backspace over the character preceding the current cursor position.
- Delete the current line.
- Delete the word at the current cursor position.

The following operations operate on blocks of text.

- Select a block of text.
- Copy the selected block of text to the clipboard.
- Cut the selected block onto the clipboard.
- Paste the contents of the clipboard into the buffer at the current cursor position

□ Text search operations

STEP supports the following text search operations.

- Find a text string in the current buffer, starting at the current cursor position.
- Find the next occurrence of the current search string, starting at the cursor position. (Forward Search)
- Find the previous occurrence of the current search string, starting at the current cursor position. (Backward search)
- Find and replace multiple occurrences of a string in the current buffer, starting at the current cursor position, with an option of replace confirmation for every occurrence.

Continued

□ Miscellaneous Operations

Other than the operations previously mentioned, STEP additionally supports the following operations.

- Page Setup (Set Left, Right, Top and Bottom Margins).
- Show Line Numbers.
- Jump to a line.
- Color Setup (Foreground and Background colors for the working area of the STEP window.
- Set beeps ON or OFF.
- Refresh the user screen.

2.3 USER CHARACTERISTICS

STEP is intended as a **general-purpose text-editing program** for use by any Linux user wishing to view or edit pure ASCII text files. Additionally programmers wishing to write their source programs, shell scripts etc. can also use STEP for their editing needs.

- ✓ STEP has been designed mainly keeping in mind the needs of users new to Linux usage and/or programming. For this reason, the design philosophy of STEP has been to keep the interface very simple and straightforward to learn and use.
- ✓ A considerable effort has been made during the overall design of STEP to keep the **learning curve** of the software as low as possible, to enable new users to progressively adapt to Linux environment, before using advanced editing systems like VIM.
- ✓ STEP also tries to create a familiar working environment for MS-DOS and WIN95/98 users. Hence its interface is quite similar to that of the popular MS-DOS editor, **EDIT**. **EDIT** users will find familiar options in the STEP menu systems. Additionally, STEP has the ability to open **DOS format** text files.

2.4 GENERAL CONSTRAINTS

The following points the general usage level constraints of STEP.

- ❑ STEP cannot work with very large files, i.e. Files greater than 100 MB in size.
- ❑ STEP cannot open or work with files for which the user does not have read permissions, due to the file security settings in LINUX.
- ❑ STEP currently does not provide a file listing for the user to choose a desired file to open.
- ❑ STEP imposes a restriction on the maximum number of characters in a line to 1024, when opening a file from disk.
- ❑ STEP also imposes a restriction on the number of open files at any given time, to a minimum of 1 and a maximum of 16.

2.5 ASSUMPTIONS & DEPENDENCIES

The following **Design Assumptions** have been made during the design of STEP.

- A. A computer with the following minimum requirements: Processing power equivalent to a 486DX or higher processor, 8 MB of RAM, and 4MB of hard disk space.
- B. An ASCII compatible keyboard, and optionally, a mouse.
- C. A functional Linux System running Redhat Linux 7.0 or higher.
- D. A functional Program execution environment i.e. A TTY device terminal running any shell command prompt interface, like bash, ksh, csh etc.

Continued

The following are the **Design Dependencies** of STEP. The presence of the following packages is necessary for the proper functioning of the software.

- I. **The GNU development libraries** - The `glibc` library of functions, and the `gcc` compiler development environment. These libraries are required for compiling the STEP sources, and additionally contain library functions used by STEP.
- II. **The `ncurses` development libraries** – For Window management and User Interface Menu management routines.

The `ncurses` package is an Application Programming Interface (API) or a subroutine library for terminal-independent screen painting and input-event handling which presents a high-level screen model to the programmer, hiding differences between terminal types. `Ncurses` uses `terminfo`, which is a database format that can describe the capabilities of thousands of different terminals

The `ncurses` library of APIs consists of the following –

- ❑ **Windows** – `ncurses` contains functions for creation, deletion and manipulation of user interface elements called windows. These windows are used for displaying text onto the screen.
 - ❑ **Menus** – The menu library is a library of functions for creation, deletion and manipulation of User Interface Menu Systems.
 - ❑ **Panels** – The `panel` library of functions allow association of a backing store with each of a stack or deck of overlapping windows, and provides operations for moving windows around in the stack that change their visibility in the natural way (handling window overlaps).
 - ❑ **Forms** - The `form` library is a `curses` extension that supports easy programming of on-screen forms for data entry and program control.
- III. **The C++ Standard Template Library** – For the vector, list and string template classes. The STL has predefined template data structure classes that can be instantiated in application programs that use this library. The buffer used to hold the textual data in STEP is a STL known as the `vector`.

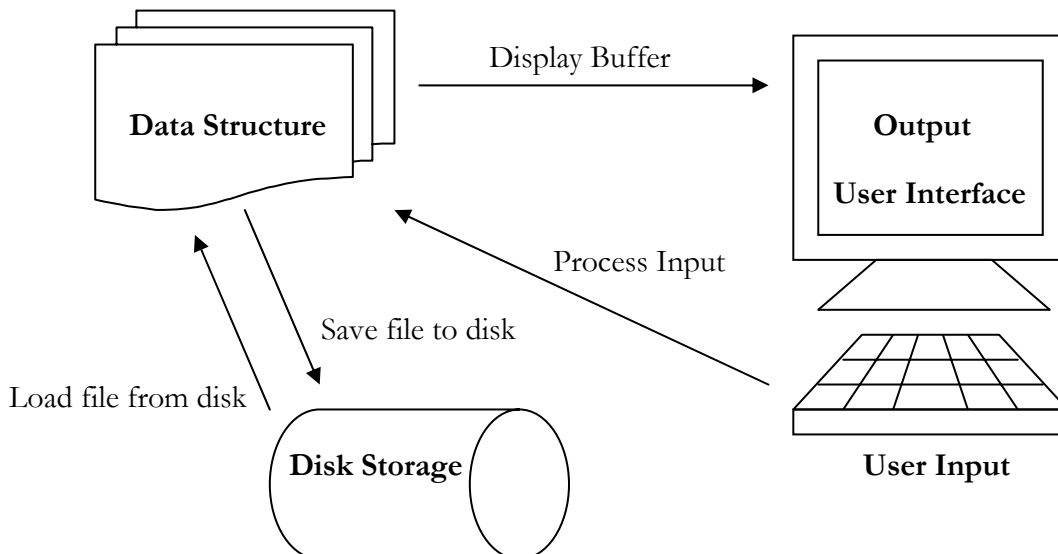
3. FUNCTIONAL REQUIREMENTS

The functional requirements and specifications of STEP are explained in detail below. The content has been divided into different sections to facilitate easy understanding of the design of the editor.

FUNCTIONAL REQUIREMENTS - I

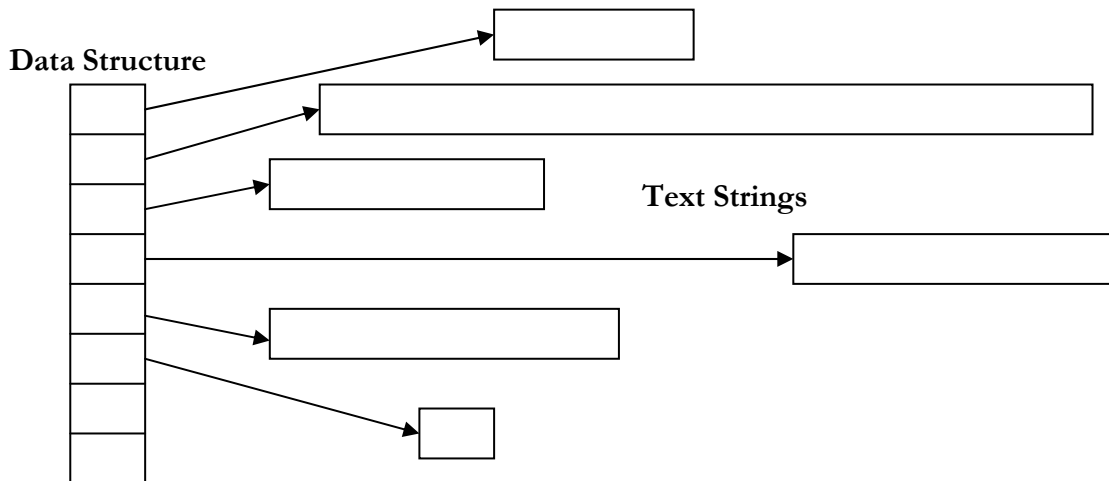
INTRODUCTION

The following diagram illustrates the basic architectural design of STEP.



BASIC ARCHITECTURAL DESIGN OF STEP

The buffer used to store text read in from the user input, and/or from loading files from the disk is a data structure, designed to hold the text in units of strings, each unit containing one line of text. This idea is illustrated in the diagram below.



THE TEXT BUFFER DATA STRUCTURE

The diagrams show the basic design of STEP. The various components of the diagrams, and the details of operation of the editor are explained in the subsections below.

INPUT

STEP accepts inputs from the user from the keyboard, and optionally from the mouse. The input to STEP can be any one of the following:

1. **An ASCII-printable character** – In this case, STEP accepts this character as text input, and inserts it into the current buffer appropriately.
2. **A Navigation command** – The user can issue navigation commands using any of the navigation keys on the keyboard, like the ARROW keys, HOME, END etc. When such a request is recognized, STEP appropriately the text cursor to fulfill the request.

Continued

3. **A Mouse button event** – STEP has the capability to process Mouse Inputs, during Menu interaction sequences. The user can select an option from the menu using the mouse
4. **A Control key sequence** – The user can also issue specialized commands to STEP, for performing specific operations, by using a combination of the CTRL (META), ALT and FUNCTION keys. When such a key sequence, known as a hotkey sequence is recognized, STEP performs the desired operation.

The input operations of STEP are interfaced with the hardware of the system by the ncurses library. The library has a large database of functions for performing input operations. These functions can be used to obtain data from the user, in the form of text input, numeric values etc. By using only these functions for performing I/O, the design of STEP is ensured to be generic across various terminal types and underlying hardware.

Disk Input

In addition to these methods input, STEP performs input operations from the disk storage, while reading files and loading into the text buffers. In this regard, the following facts are relevant.

- STEP can open files stored at any location in any of the currently mounted file systems.
- STEP cannot open files for which the user does not have read permissions. In such a situation, STEP reports an error message to the user.
- STEP can open files on the disk that are of the DOS text format.
- STEP can have multiple files open at any time. Each file is associated with its own buffer. STEP allows users to perform text block operations between the open buffers.

PROCESSING

The basic approach to input processing in STEP is fundamental to **conditional switching** and program execution according to the precepts of **structural programming**. When STEP is first loaded into memory as a process by the shell, it execute some initialization code, and enters an input processing loop structure, where it waits for input from the user, and processes it appropriately, using a conditional switching logic.

The basic algorithm for accomplishing this is illustrated below. This sequence of instructions is endlessly and repeatedly executed till the user explicitly terminates the loop by issuing an exit command.

```
While (not terminated) {  
    Get input from the user  
    Determine the type of the input character {  
        If it is an ASCII printable character, insert into  
        the current buffer.  
  
        If it is a navigation command, appropriately  
        position the cursor.  
  
        If it is a menu command sequence, execute the menu  
        operation requested.  
  
        If it is a control key sequence, execute the  
        requested command.  
  
        If it is a mouse event, process the request.  
  
        If it is none of the above cases, ignore it.  
  
        If it is an exit command, execute pre-termination  
        code, and exit the loop.  
    }  
}
```

THE MAIN INPUT-PROCESSING LOOP

OUTPUT

The output from STEP, occurring as a response to user requests, is directed towards two destinations. These are –

- **The User's terminal** – STEP outputs messages, status information, Error messages, and other diagnostic messages to the user via his/her terminal interface. This interface is completely managed by the `ncurses` API library. STEP issues requests, in the form of functions, to `ncurses`, which in turn produces recognizable output at the terminal.

The `ncurses` library uses window systems; onto output information is depicted to the user. Windows are demarcated areas of the user's terminal, onto which programs using the `ncurses` library can output text and numeric information. Additionally, the window output can be formatted as desired by the application programmer. STEP uses `ncurses` windows to display its text buffers to the user.

- **The disk storage** – STEP also outputs data to the disk storage, normally the hard disk, as a response to save requests by the user. When such a request occurs, STEP writes the contents of the text buffer to disk.

STEP can save the contents of its buffers to a file at any location in any of the currently mounted file systems. This gives the user a large amount of flexibility in saving the file to any desired location. In case there already exists a file with the same pathname and filename as the specified one, STEP prompts the user whether the existing file should be replaced. If the user does not have permissions to write to the existing file or directory, STEP prints an appropriate error message.

FUNCTIONAL REQUIREMENTS - II

SOURCE CODE ORGANIZATION

The STEP sources are organized into the following files. The code has been separated into these separate files according to content, and functionality.

- ❑ **step_doc.h** – This file contains the class definition for the main text container class, and the global declarations.
- ❑ **step_menu.h** – This file contains the class definition for the menu object class.
- ❑ **step_main.cpp** – This file contains the main() function, the input processing function, and other miscellaneous user interface functions.

CLASS DEFINITIONS

The class definitions in the STEP sources are detailed below:

The **text_document** class

The **text_document** class is the main text buffer class. It contains the data structure and associated data members necessary to store, manipulate and retrieve text data. It also contains methods, or member functions, which perform operations on the data attributes. By keeping the visibility of the attributes, **data encapsulation** and **data hiding** is implemented.

The important data members of this class are:

text_buf - A buffer of strings to hold the textual information as strings

cur_line, cur_col - Line and column indicators to the current position of the cursor.

top_line, bot_line- Indicator to the first and last lines currently being displayed

text_win - Pointer to the associated display window, and its size parameter variables.

is_modified - Flag to indicate whether the buffer has been modified since the last save operation.

Continued

The important member functions of this class are listed below:

text_document() - The text_document class constructor - initializes the data members, and optionally the buffer, with the contents of a file. It also creates the associated window for displaying the buffer.

~text_document() - The text_document class destructor - Deletes dynamically allocated data and executes cleanup code.

insert_text() - Inserts specified text into the current buffer at the specified position.

delete_text() - Deletes specified text in a specified line of buffer.

insert_line() - Inserts a specified line at the specified position into the current buffer.

delete_line() - Deletes specified line from the current buffer.

show_text() - Displays the currently visible portion of the current buffer onto its associated window.

move_left(), move_right(), move_up(), move_down(), move_home(), move_end(), page_up(), page_down() - Functions that move the cursor to the specified position.

page_up(), page_down(), scroll_up(), scroll_down() - Functions for moving around in the current buffer.

load_file() - Loads a specified file into the current buffer.

save_file() - Saves the current buffer onto disk in a specified file.

find_string() - Searches for a specified text string, starting at the current cursor position.

find_next(), find_prev() - Find the next or previous occurrence of the pre-specified search string.

find_and_replace() - Finds and replaces occurrences of the specified text string.

cut_text() copy_text() paste_text() - Functions for performing block text operations.

Continued

The `menu_obj` class

The `menu_obj` class contains attributes for defining a menu object. It also contains methods for displaying the menu object instances to the user and managing the menu interface.

The important data members of this class are:

`menu_items` – Array of menu item name and description fields.

`menu_ptr` – Pointer to menu data structure.

`menu_win, menu_sub, menu_shadow` – Windows to display the menu onto the screen.

`menu_name` – Buffer to hold the name of the menu.

`item_count` – Count of the number of items currently in the menu.

`ypos xpos` – Screen position at which to display the menu.

The important methods of this class are listed below.

`menu_obj()` – Constructor to initialize the data members of the `menu_obj` class.

`~menu_obj()` – Destructor for the `menu_obj` class.

`next_item(), prev_item()` – Moves to the next/previous item in a menu.

`show()` – Shows the currently active menu on the screen, in its associated windows.

`hide()` – Removes the currently displayed menu from the screen.

`process_item()` – Contains code for processing a menu option selected by the user.

Additionally, the STEP sources contain the following user interface functions.

`printmsg()` – Prints a simple message at the bottom of the user screen.

`msgbox()` – Displays a message window with a bi-level choice to the user.

`inputbox()` – Accepts variable data-type input from the user.

`textbox()` – Displays specified text in a window.

4. EXTERNAL INTERFACE REQUIREMENTS

The interface that STEP maintains with its environment is explained in detail in the following subsections. These methods form the fundamentals of commonly used interface systems, and present a familiar and easily adaptable environment to the user.

USER INTERFACE

The user interface offered by STEP is a very simple and easy to use one. The learning curve for becoming familiar with the STEP User Interface is negligible. This ease of use is made possible by using the following user interfacing methods –

Dialog Boxes – STEP incorporates the concept of dialog boxes, found in many interactive software systems. Dialog boxes provide an easy-to-understand and efficient method of communication between the user and the software. The following types of Dialog box systems are used in STEP.

- ❑ **Message Box** – This type of dialog box is used to present a prompt with a title to the user, and give an option to answer to the prompt in the affirmative or the negative, by selecting one of the available choices.
- ❑ **Input Box** – The Input Box dialog system is used for accepting input from the user. The input data can be of string, integer or other data-type formats. A dialog box is presented to the user, with a field for entering the requested information. Also, a choice of options is provided so that the operation can be canceled at any time.
- ❑ **Text Box** – The Text Box dialog box is a simple one, and it presents a simple text message to the user. It does not have the facility to ask any questions or return any information to the program. It is mainly used to display relatively large amounts of text information to the user.

Continued

Menu interface systems - STEP uses pull-down menu systems to present the user with a set of selectable choices. Additionally, for experienced users, STEP also offers keyboard shortcuts to most of the STEP menu options, to enable them to work faster. The Menu systems implemented in STEP are detailed below.

- **File Menu** – The file menu contains options for performing various file operations.
 - **New** – Creates a new text document.
 - **Open** – Creates a new text document and loads it with a file from disk. The user is prompted to enter desired file to open.
 - **Close** – Closes the currently active document. The user is prompted to save the document, if unsaved.
 - **Save** – Saves the currently active document.
 - **Save as** – Saves the currently active document with a different name.
 - **Delete** – Deletes a specified file from the disk
 - **Insert** – Inserts a specified file from disk after the current line position in the currently active document
 - **Exit** – Exits STEP. The user is prompted to save any unsaved files to disk.

- **Edit menu** – The Edit menu contains options for performing text block operations.
 - **Set Begin Mark** – Sets the beginning mark for a block of text.
 - **Cut** – Cuts out the selected block of text, starting at the begin mark position, and terminating at the current cursor position, and puts it into the clipboard.
 - **Copy** – Copies the selected block of text, starting at the begin mark position, and terminating at the current cursor position, and puts it into the clipboard.
 - **Paste** – Pastes the contents of the clipboard into the currently active document at the current cursor position.

Continued

- **Options Menu** – The Options menu contains settings and preferences that the user might like to modify.
 - **Show Line Numbers** – Displays line numbers next to each line in the file.
 - **Go to Line** – Jumps to a specified line in the currently active document.
 - **Beeps On/Off** – Toggles the Beeps sounded by STEP to indicate the cursor position.
 - **Page Setup** – Allows the user to set margins for the current document.
 - **Color Setup** – Allows the user to select a desired foreground-background color combination for the working area of the STEP window.
 - **Refresh** – Refreshes the STEP user area

- **Tools Menu** – The tools menu contains tools for performing specialized operations on current text document.
 - **Find** – Searches for a specified string in current document, starting at the current cursor position. The user is prompted to enter the search string.
 - **Find Next** – Finds the next occurrence (if any) of the current search string. (Forward Search)
 - **Find Previous** – Finds the previous occurrence (if any) of the current search string.
 - **Find and Replace** – Finds and replaces multiple occurrences of the specified string in the current document, with an option of confirming each replacement.

- **Help Menu** – The help menu contains options to display STEP user commands and information about STEP.
 - **Help** – Displays a text box enumerating the commands in STEP.
 - **About** – Displays a text box with information about STEP.

HARDWARE INTERFACE

The hardware requirements required by STEP for its proper functioning are very minimal. These basic requirements are listed below.

- ✓ A minimum of 486DX processor or equivalent.
- ✓ 4 MB of free user memory.
- ✓ 10 MB of free hard disk space.
- ✓ An ASCII compatible keyboard, an optionally, a PS/2 or serial interface mouse.
- ✓ A hardware terminal interface.

SOFTWARE INTERFACE

The overall software interface required by STEP consists of the following components:

- ✓ A functional Redhat Linux 7.0 or higher system.
- ✓ The **ncurses** software API library – is for window management and terminal interfacing. The **new curses** library forms the generic interface layer between STEP and the underlying hardware, allowing the STEP design to be hardware independent.
- ✓ The **GNU C++** and **gcc** compiler environment – the **glibc** compiler environment contains a large base of general APIs for performing various functions.
- ✓ The C++ Standard Template Library - The C++ STL templates are used for management of the data structures in STEP, and hence are essential to the proper execution of STEP. These templates form the basic structure of text buffer databases in STEP.

5. PERFORMANCE REQUIREMENTS

The performance requirements for STEP can be stipulated as follows:

1. **Ability to manage large files** – STEP can comfortably work with files with sizes of 20 MB or less.
2. **Robustness** – STEP handles most of the possible error conditions while that can occur while processing, like the ones listed below.
 - a. **Corrupted files** and files with **non-ASCII printable content** - When loading such files, STEP reports an error message and aborts the file loading process.
 - b. **Memory allocation errors** – STEP is equipped to handle memory allocation failures that may occur due to insufficient memory space.
 - c. **File system errors** – STEP also handles abnormal situations occurring due to file operation errors due to errors in the file system.

6. DESIGN CONSTRAINTS

The following constraints apply to STEP operations due to its STEP.

- STEP cannot work with extremely large files (Size greater than 50 MB).
- STEP cannot process non-ASCII information.
- Due to its interactive nature, STEP cannot operate in batch processing environment.
- STEP cannot open files with extremely long lines with 1024 characters or more.

STANDARD COMPLIANCE

STEP is currently compatible with the following standards. Its performance heavily depends on the availability of these standards in its working environment.

- ✓ STEP is compliant only with the **AT&T UNIX System V** release of the `ncurses` library.
- ✓ STEP is designed to comply with the **ANSI C++ Standard Template Library** implementation.
- ✓ STEP has been designed and tested only for the **GNU gcc compiler** environment. Its performance in any other environment is yet untested.
- ✓ Due to its dependence on `ncurses`, STEP is compliant only with standard character-cell terminal interfaces.

HARDWARE LIMITATIONS

The hardware limitations that affect the functionality of STEP are listed below.

- STEP can work only on systems with hardware terminal interfaces compatible with the underlying `ncurses` library.
- The STEP programming logic requires an ASCII standard compatible keyboard interface to accept user input. Any other standard has not been implemented in STEP.

APPENDIX

SOURCE CODE LISTING OF STEP

File #1: step_doc.h

```
#include <string>
#include <fstream>
#include <vector>
#include <unistd.h>
#include <math.h>
#include <signal.h>
#include <time.h>
#include <menu.h>

#define TAB_WIDTH 8 /* the tab width */
#define MAX_LEN 256 /* max filename size */
#define LINE_SIZE 1024 /* max line size */

/* the text_document class */
class text_document {
    vector<string *> text_buf; /* the text buffer */
    long int top_line, bot_line; /* top and bot line indexes respectively */
    long int search_line; /* search string index */
    int first_char; /* col pos of first char of topmost line in cur
screen */
    int max_col; /* value of max column position reached */
    int ypos, xpos; /* current y,x coordinates of cursor */
    int left_mar, right_mar;
    int top_mar, bot_mar; /* margin values at the four sides */
    int file_id, line_num; /* file index and line number count */
    int search_pos;
        /* search string column pos */
    string search_string; /* current search string */

    void scroll_up();
    void scroll_down();
    bool load_file(const string&);
    void empty_buf();
    inline string* new_line(const string&);
```

```
public:
WINDOW *text_win, *bor_win;    /* the associated window ptrs */
long int cur_line;
int cur_col;                   /* iter to current line */
int width, height, max_y;     /* window size variables */
bool is_modified;
    /* flag to indicate modification status of current buffer */
long int beg_line;
int beg_col;                   /* begin marker (iter,int) pairs */
string file_path;
    /* path to the file (if any) associated with current
buffer */

int file_index(void) { return(file_id); }
int num_lines() { return(text_buf.size()); }
inline int untab_col(const int, const string&);
inline int wrap_lines(const string&);
inline int line_ypos(int);
inline int tab_col(const int, const string&);
inline const string& get_line(int);

text_document(const string&);
text_document(int, int);
~text_document();
bool is_empty();
bool save_file(string);
int gotoline(int);

bool insert_file(const string&, int);
void insert_line(int, const string&);
void insert_text(const string&, int, int);
void delete_line(int);
void delete_text(int, int, int);

int move_left();
int move_right();
int move_up();
int move_down();
void move_home();
void move_end();
int page_up();
int page_down();

bool find_string(const string&, int, int, bool);
int find_next(int, int);
int find_prev(int, int);
int find_and_replace(string, string, bool);

void cut_text();
bool copy_text();
void paste_text();
```

```

void set_left_mar(int);
void set_right_mar(int);
void set_beg_mar(int);
void set_end_mar(int);

void show_line_num();
void switch_doc();
void show_text();
void print_status();
};

/* some global declarations */
bool beeps_on=TRUE, ins_mode_on=TRUE; /* some bool switches */
vector<text_document *> doc_vec; /* text document list */
int doc_index;
text_document *cur_doc; /* current doc iter */
text_document clipboard(LINES-4, COLS-2);
/* the clipboard buffer */
void printmsg(char*,...);
bool msgbox(char* prompt, char* format,...);
bool inputbox(char*, char*, void*);
void process_signal(int);

/* the text document class member function definitions */

/* get_line - returns a specified line string from the text buffer.
params: line_num - required zero index line number.
return: the desired string
*/
const string& text_document::get_line(int line_num=cur_doc->cur_line) {
    return(*text_buf[line_num]);
}

/* new_line - creates a new string, and does error processing.
params: line - content to initialize the new string with.
return: ptr to newly created string.
*/
string* text_document::new_line(const string& line) {
    string* temp_strp=new string(line);
    if(!temp_strp)
        process_signal(0);
    return(temp_strp);
}

/* tab_col - func to return column pos of a char in a line,
accounting for tab spaces.
params: col - a char position
str - a string to process
return: the column pos with the tab spaces.
*/

```

```

int text_document::tab_col(const int col=cur_doc->cur_col,
                          const string& str=cur_doc->get_line()) {
    int xpos=0,line;
    for(int i=0;i<col;i++) {
        line=xpos/width;
        if(str[i]=='\t') {
            xpos+=TAB_WIDTH-(xpos+(COLS-width)*line)%TAB_WIDTH;
            if(xpos/width!=line)
                xpos=(line+1)*width;
        }
        else
            xpos++;
    }
    return(xpos);
}

/* untab_col - func to return column pos of a char in a line,
   removing tab spaces. (reverse of tab_col())
params: xpos - a col position in a line
       str - the string to process
return: the column position with tab spaces removed.
*/
int text_document::untab_col(const int xpos,
                             const string& str=cur_doc->get_line()) {
    int col=0,line;
    for(int i=0;i<xpos;i) {
        line=i/width;
        if(str[col]=='\t') {
            i+=TAB_WIDTH-(i+(COLS-width)*line)%TAB_WIDTH;
            if(i/width!=line)
                i=(line+1)*width;
        }
        else
            i++;
        col++;
    }
    return(col);
}

/* wrap_lines - func to return the number of physical lines
   that a logical line wraps to.
params: line - the line to process.
return: the integer number of lines that 'line' wraps to.
*/
int text_document::wrap_lines(const string& line=cur_doc->get_line()) {
    int new_size=tab_col(line.size(),line);
    if(new_size<width)
        return(1);
    else return(new_size/width+1);
}

```

```

/* line_ypos - func to return y position of the first character of a
line.
params: line_iter - the line to find the ypos of.
return: the ypos of the first character of the line.
*/
int text_document::line_ypos(int line_iter=cur_doc->cur_line) {
    if(line_iter<top_line)
        return(-1);
    else if(line_iter==top_line)
        return(0);
    int i=wrap_lines(text_buf[top_line]->
substr(untab_col(first_char,get_line(top_line)))));
    int iter=top_line+1;
    for(;iter!=line_iter;iter++)
        i+=wrap_lines(get_line(iter));
    return(i);
}

/* the text_document class destructor - empties the text buffers and
deletes associated windows.
*/
text_document::~~text_document() {
    empty_buf();
    if(text_win)
        delwin(text_win);
    if(bor_win)
        delwin(bor_win);
}

/* insert_line - func to create a new line in the current buffer.
params: line_iter - iter pointing to position before which
line is to be inserted.
return: nothing.
*/
void text_document::insert_line(
int line_iter=cur_doc->cur_line,const string&
line="") {
    text_buf.insert(text_buf.begin()+line_iter,new_line(line));
}

/* delete_line - deletes a specified line from the text buffer.
params: line_iter - index to the line to be deleted.
return: nothing.
*/
void text_document::delete_line(int line_iter=cur_doc->cur_line) {
    if(is_empty())
        return;
    if(text_buf.size()==1) {
        text_buf[0]->erase(0,text_buf[0]->size());
        return;
    }
}

```

```

delete(text_buf[line_iter]);
text_buf.erase(text_buf.begin()+line_iter);
}

/* insert_text - inserts specified text into specified line at specified
postion.
params: insert_str - string to be inserted.
       line_pos - line into which insert_str is to be inserted.
       col_pos - col pos in line_pos at which insert_str is to be
inserted.
return: nothing.
*/
void text_document::insert_text(const string& insert_str,
                               int line_pos=cur_doc->cur_line,int col_pos=cur_doc->cur_col)
{
    text_buf[line_pos]->insert(col_pos,insert_str);
}

/* delete_text - deletes text specified by line, postion and length.
params: count - number of characters to be deleted.
       line_pos - index of line in which text has to be deleted.
       col_pos - col pos in line_pos at which deletion has to begin.
return: nothing.
*/
void text_document::delete_text(int count=1,
                                int line_pos=cur_doc->cur_line,int col_pos=cur_doc->cur_col) {
    text_buf[line_pos]->erase(col_pos,count);
}

/* empty_buf - func to empty the current buffer.
params: nothing.
return: nothing.
*/
void text_document::empty_buf() {
    for(int i=0;i<text_buf.size();i++)
        if(text_buf[i])
            delete(text_buf[i]);
    text_buf.clear();
    top_line=first_char=cur_line=0;
}

/* scroll_up - func to scroll up one physical line,
in the associate window and in the current buffer.
params: nothing
return: nothing
*/

```

```

void text_document::scroll_up() {
    if(tab_col(text_buf[top_line]->size(),get_line(top_line)) < width ||
        (first_char==width)<0)
        --top_line,

    first_char=(tab_col(get_line(top_line).size(),get_line(top_line))/width
)*width;
    show_text();
}

/* scroll_down - func to scroll down one physical line,
   in the associate window and in the current buffer.
params: nothing
return: nothing
*/
void text_document::scroll_down() {
    int new_size=tab_col(text_buf[top_line]->size(),get_line(top_line));
    if(new_size < width || (first_char+=width) > new_size)
        ++top_line,first_char=0;
    show_text();
}

/* move_right - func to move right one logical position,
   in the associated window and in the current buffer.
params: nothing
return: 1 on successful move, 0 on failure.
*/
int text_document::move_right() {
    int newx;
    getyx(text_win,ypos,xpos);
    if(!text_buf[cur_line]->size() || cur_col==text_buf[cur_line]->size())
    {
        if(beeps_on) beep();
        return(-1);
    }
    ++cur_col; max_col=tab_col();
    newx=max_col%width;
    if(ypos==max_y && newx<xpos)
        scroll_down();
    else
        ypos+=(newx<xpos && ypos<max_y?1:0);
    xpos=newx;
    wmove(text_win,ypos,xpos);
    return(0);
}

/* move_left - func to move left one logical position,
   in the associated window and in the current buffer.
params: nothing
return: 1 on successful move, 0 on failure.
*/
int text_document::move_left() {
    int newx;
    getyx(text_win,ypos,xpos);

```

```

if(!text_buf[cur_line]->size() || !cur_col) {
    if(beeps_on) beep();
    return(-1);
}
--cur_col; max_col=tab_col();
newx=max_col%width;
if(ypos==0 && newx>xpos)
    scroll_up();
else
    ypos--=(newx>xpos && ypos>0?1:0);
xpos=newx;
wmove(text_win,ypos,xpos);
return(0);
}

/* move_up - func to move up one logical line,
   in the associated window and in the current buffer.
params: nothing
return: 1 on successful move, 0 on failure.
*/
int text_document::move_up() {
    int wrap,line_y;
    getyx(text_win,ypos,xpos);
    line_y=line_ypos();
    if(!cur_line) {
        if(beeps_on) beep();
        return(-1);
    }
    else {
        --cur_line;
        wrap=wrap_lines();

        if(max_col>tab_col(text_buf[cur_line]->size()))
            xpos=tab_col(cur_col=text_buf[cur_line]->size());
        else
            xpos=max_col, cur_col=untab_col(max_col);

        if(wrap>height) {
            top_line=cur_line, first_char=cur_col=0;
            wmove(text_win,0,0);
            show_text();
        }
        else if(line_y-wrap<0) {
            for(int i=0;i<(wrap-line_y);i++)
                scroll_up();
            wmove(text_win,xpos/width,xpos%width);
        }
        else
            wmove(text_win,line_y-wrap+xpos/width,xpos%width);
    }
    return(0);
}
}

```

```

/* move_down - func to move down one logical line,
   in the associated window and in the current buffer.
params: nothing
return: 1 on successful move, 0 on failure.
*/
int text_document::move_down() {
    int line_y,wrap;
    getyx(text_win,ypos,xpos);
    if(!num_lines() || cur_line==num_lines()-1) {
        if(beeps_on) beep();
        return(-1);
    }
    else {
        cur_line++;
        line_y=line_ypos();

        if(max_col>tab_col(text_buf[cur_line]->size()))
            xpos=tab_col(cur_col=text_buf[cur_line]->size());
        else
            xpos=max_col, cur_col=untab_col(max_col);
        wrap=wrap_lines();
        if(wrap>height) {
            top_line=cur_line, first_char=cur_col=0;
            wmove(text_win,0,0);
            show_text();
        }
        else if((line_y+wrap)>height) {
            for(int i=0;i<(line_y+wrap-height);i++)
                scroll_down();
            wmove(text_win,height-wrap+xpos/width,xpos%width);
        }
        else
            wmove(text_win,line_y+xpos/width,xpos%width);
    }
    return(0);
}

/* move_end - func to move to the end of cur logical line,
   in the associated window and in current buffer.
params: nothing
return: nothing
*/
void text_document::move_end() {
    int line_y,wrap;
    cur_col=text_buf[cur_line]->size();
    max_col=tab_col(cur_col);
    wrap=wrap_lines(); line_y=line_ypos();

    if(wrap>height) {
        for(int i=0;i<wrap-height;i++)
            scroll_down();
        wmove(text_win,max_y,max_col%width);
    }
}

```

```

else if(line_y+wrap>height) {
    for(int i=0;i<line_y+wrap-height;i++)
        scroll_down();
    wmove(text_win,max_y,max_col%width);
}
else
    wmove(text_win,line_ypos(cur_line)+max_col/width,max_col%width);
}

/* move_home - func to move to the beg of cur logical line,
   in the associated window and in the current buffer.
params: nothing
return: nothing
*/
void text_document::move_home() {
    max_col=cur_col=0;
    int wrap=wrap_lines();
    if(wrap>height) {
        for(int i=0;i<wrap-height;i++)
            scroll_up();
        wmove(text_win,0,0);
    }
    else
        wmove(text_win,line_ypos(cur_line),0);
}

/* page_up - func to move up one page,
   in the associate window and in the current buffer.
params: nothing
return: nothing
*/
int text_document::page_up() {
    if(top_line<=0) {
        if(beeps_on) beep();
        return(-1);
    }
    for(int i=0;i<max_y && top_line!=0;) {
        --top_line; i+=wrap_lines(get_line(top_line));
    }
    first_char=0,cur_col=0; cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
    return(0);
}

/* page_down - func to move down one page,
   in the associated window and in the current buffer.
params: nothing
return: nothing
*/

```

```

int text_document::page_down() {
    if(bot_line==text_buf.size()-1) {
        if(beeps_on) beep();
        return(-1);
    }
    for(int i=0;i<max_y && top_line!=text_buf.size()-1;) {
        ++top_line; i+=wrap_lines(get_line(top_line));
    }
    first_char=cur_col=0; cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
    return(0);
}

/* show_text - func to show currently active section of current buffer
   onto associated window.
params: nothing
return: nothing
*/
void text_document::show_text() {
    int y,x,cur_x,cur_y;
    getyx(text_win,cur_y,cur_x);
    werase(text_win);
    if(line_num) {
        for(int i=1;i<=height+top_mar+bot_mar;i++)
            mvwaddstr(bor_win,i,1," ");
        mvwprintw(bor_win,top_mar+1,1,"%d",top_line+1);
    }
    mvwaddstr(text_win,0,0,(text_buf[top_line]->substr(
untab_col(first_char,get_line(top_line))))).c_str());
    getyx(text_win,y,x);
    bot_line=top_line; bot_line++;
    for(;bot_line!=text_buf.size();bot_line++) {
        if(line_num)
            mvwprintw(bor_win,top_mar+y+2,1,"%d",bot_line+1);
        mvwaddnstr(text_win,y+1,0,text_buf[bot_line]->c_str(),(max_y-
y)*width);
        if(text_buf[bot_line]->size()>(max_y-y)*width)
            bot_line--;
        getyx(text_win,y,x);
        if(y==max_y)
            break;
    }
    if(bot_line==text_buf.size())
        bot_line--;
    wnoutrefresh(bor_win);
    wmove(text_win,cur_y,cur_x);
}

```

```

/* load_file - func to load a file from disk onto the current buffer.
params: nothing
return: 1 if file exists, 0 if file does not exist.
note: this func also does processing for properly opening dos text
files.
*/

```

```

bool text_document::load_file(const string& filename) {
    char line_buf[LINE_SIZE];
    ifstream infile;
    printmsg("Status: Opening file \"%s\"...",filename.c_str());
    douupdate();
    infile.open(filename.c_str(),ios::in);
    if(infile) {
        while(infile.getline(line_buf,LINE_SIZE))
            text_buf.push_back(new_line(line_buf));
        if(!infile.eof()) {
            msgbox("File Open Error!!","Line size too big!!.");
            printmsg("File open error!!.");
            empty_buf();
        }
        else
            printmsg("Current File: %s [%d
line(s)].",filename.c_str(),num_lines());
        if(!num_lines())
            insert_line(0,"");
        infile.close();
        if(text_buf[0]->c_str()[text_buf[0]->size()-1]=='\r')
            for(top_line=0;top_line!=text_buf.size();top_line++)
                text_buf[top_line]->erase(text_buf[top_line]->size()-1);
        return(true);
    }
    printmsg("New file.");
    return(false);
}

```

```

/* insert_file - opens and inserts a file into the current buffer,
after a specified line position.

```

```

params: filename - name of file to insert.

```

```

        line_pos - index of line after which the file has to be
inserted.

```

```

return: TRUE if file inserted successfully, else FALSE

```

```

*/

```

```

bool text_document::insert_file(const string& filename,
                                int line_pos=cur_doc->cur_line) {
    char line_buf[LINE_SIZE];
    ifstream infile;
    printmsg("Status: Opening file \"%s\"...",filename.c_str());
    douupdate();
    infile.open(filename.c_str(),ios::in);
    if(infile) {
        for(int i=cur_line+1;infile.getline(line_buf,LINE_SIZE);i++)
            insert_line(i,string(line_buf));
        if(!infile.eof()) {
            msgbox("File Open Error!!","Line size too big!!.");

```

```

    printmsg("File open error!!.");
}
else
    printmsg("Success: File \"%s\" inserted after line %d.",
filename.c_str(),cur_line+1);
    show_text();
    return(TRUE);
}
return(FALSE);
printmsg("Error: Could not insert file \"%s\" into current buffer.",
filename.c_str());
}

/* text_document class constructor - initializes all data members,
associated windows, and optionally loads a file into the text buffer.
*/
text_document::text_document(const string& filename=""):
line_num(0),left_mar(0),right_mar(0),top_mar(0),bot_mar(0),search_strin
g(""),
search_line(-1),search_pos(-1),file_id(0),beg_line(-1),max_col(0),
height(LINES-4),width(COLS-2),max_y(height-1) {

    static int doc_num=1;
    bor_win=newwin(LINES-2,COLS,1,0);
    text_win=newwin(height,width,2,1);
    wbkgd(bor_win,' '|COLOR_PAIR(1));
    wbkgd(text_win,' '|COLOR_PAIR(1));
    nodelay(bor_win,TRUE);
    keypad(bor_win,TRUE); keypad(text_win,true);
    wborder(bor_win,
ACS_VLINE|COLOR_PAIR(2)|A_REVERSE,ACS_VLINE|COLOR_PAIR(2)|A_REVERSE,
ACS_HLINE|COLOR_PAIR(2)|A_REVERSE,ACS_HLINE|COLOR_PAIR(2)|A_REVERSE,
ACS_ULCORNER|COLOR_PAIR(2)|A_REVERSE,ACS_URCORNER|COLOR_PAIR(2)|A_REVER
SE,
ACS_LLCORNER|COLOR_PAIR(2)|A_REVERSE,ACS_LRCORNER|COLOR_PAIR(2)|A_REVER
SE);
    wnoutrefresh(bor_win); wnoutrefresh(text_win); douupdate();

    top_line=cur_line=first_char=cur_col=is_modified=0;
    file_path=filename;
    if(file_path.empty())
        file_id=doc_num++, insert_line(0,"");
    else if(!load_file(file_path))
        insert_line(0,"");
    if(file_id) {
        mvwprintw(bor_win,0,(COLS-3)/2-5," Untitled%d ",file_id);
        mvwchgat(bor_win,0,(COLS-3)/2-
5,11,A_NORMAL|COLOR_PAIR(6),COLOR_PAIR(6),NULL);
    }
    else {
        mvwaddch(bor_win,0,(COLS-3)/2-file_path.size()/2-1,' ');
        waddstr(bor_win,file_path.c_str());

```

```

    waddch(bor_win, ' ');
    mvwchgat(bor_win,0,(COLS-3)/2-file_path.size()/2-1,
             file_path.size()+2,A_NORMAL|COLOR_PAIR(6),COLOR_PAIR(6),NULL);
}
wnoutrefresh(bor_win); douupdate();
}

/* text_document class constructor - does not initialize any
   associated windows. Also, does not load any file into the text
   buffer.
*/
text_document::text_document(int hgt, int wdt):
line_num(0),left_mar(0),right_mar(0),top_mar(0),bot_mar(0),search_string(""),
search_line(-1),search_pos(-1),file_id(0),beg_line(-1),max_col(0),
height(hgt),width(wdt),max_y(height-1),bor_win(NULL),text_win(NULL) {
    top_line=cur_line=first_char=cur_col=is_modified=0;
    insert_line(0,"");
}

/* save_file - func to save the current buffer to a file.
   params: filename - file to the save the current buffer as.
   return: false if successful, true otherwise
*/
bool text_document::save_file(string file_name=cur_doc->file_path) {
    if((file_id || file_name!=file_path) &&
!access(file_name.c_str(),F_OK)) {
        if(!msgbox("Warning","File already exists. Overwrite ?: ")) {
            printmsg("Aborted: File \"%s\" not overwritten.",file_name.c_str());
            return(true);
        }
    }
    ofstream outfile;
    outfile.open(file_name.c_str(),ios::out);
    int iter;
    if(outfile) {
        for(iter=0;iter!=text_buf.size();iter++)
            if(!(outfile<<text_buf[iter]->c_str()<<"\n"))
                break;
        outfile.close();
        printmsg("Success: Wrote file: \"%s\".",file_name.c_str());
        return(false);
    }
    msgbox("Warning","Error writing file \"%s\".", file_name.c_str());
    return(true);
}

/* is_empty - func to check whether the current buffer is 'empty'.
   params: nothing
   return: 1 if empty, 0 otherwise.
   note: the current buffer is never really empty. it contains atleast
   one(empty) line.
*/

```

```

bool text_document::is_empty() {
    return(text_buf.size()==1 && text_buf[0]->empty());
}

/* gotoline - func to jump to a specified line.
params: line_no - line number of the line to jump to.
return: -1 if unsuccessful, 0 if successful.
*/
int text_document::gotoline(int line_no=cur_doc->cur_line+1) {
    line_no--;
    if(line_no<0 || line_no>num_lines()) {
        msgbox("Error","Line number out of limits.");
        return(-1);
    }
    first_char=cur_col=0; cur_line=line_no;
    int wrap_val=0;
    for(top_line=0;top_line<line_no;top_line++)
        wrap_val+=wrap_lines(get_line(top_line));
    for(;wrap_val%height!=0;--top_line,wrap_val-
=wrap_lines(get_line(top_line)));
    show_text();
    wmove(text_win,line_ypos(cur_line),0);
    return(0);
}

/* find_string - func to find a specified string in the current buffer,
starting at the current cursor position.
params: search_str - text string to search for.
        move - switch to indicate whether to move the cursor
        to the search string position if it is found.
        wrap - switch to indicate whether or not to wrap around end of
buffer.
return: true if string is found, false otherwise.
note: this func sets the variables search_string, search_line and
search_pos
      if the search string is found.
*/
bool text_document::find_string(const string& search_str,
    int start_iter=cur_doc->cur_line,
    int start_pos=cur_doc->cur_col,bool wrap_top=TRUE) {
    int end_iter,choice=string::npos;
    end_iter=wrap_top?start_iter:num_lines()-1;
    search_string=search_str; search_line=start_iter;
search_pos=start_pos;
    if(search_pos>=text_buf[search_line]->size())
        search_pos=text_buf[search_line]->size();
    if((choice=text_buf[search_line]->
substr(search_pos).find(search_string))==string::npos) {
        search_pos=0; search_line++;
        while(search_line!=end_iter) {
            if(search_line==text_buf.size()) {
                printmsg("Search wrapped around BOTTOM of text, continuing at
TOP.");
            }

```

```

        search_line=-1;
    }
    else if((choice=text_buf[search_line]-
>find(search_string))!=string::npos)
        break;
    search_line++;
}
if(choice==string::npos)
    choice=text_buf[search_line]->find(search_string);
}
if(choice==string::npos) {
    search_string="";
    search_line=search_pos=-1;
    return(FALSE);
}
gotoline(search_line+1);
search_pos+=choice;
cur_line=search_line; cur_col=(int)search_pos;
ypos=line_ypos(); xpos=tab_col();
wmove(text_win,ypos+xpos/width,xpos%width);
getyx(text_win,ypos,xpos);
wchgat(text_win,search_string.size(),A_REVERSE,COLOR_PAIR(1),NULL);
if(xpos+search_string.size()>width) {
    mvwchgat(text_win,ypos+1,0,search_string.size()-(width-xpos),
A_REVERSE,COLOR_PAIR(1),NULL);
    wmove(text_win,ypos,xpos);
}
return(TRUE);
}

/* find_next - func to find the next occurrence (if any) of the
previously set
'search_string' variable.( forward search )
params: start position (line,col) pair.
return: -1 if no search string, 1 if found, 0 if not found.

note: 1. this func can be used only after 'find_string' has located
an occurrence of the search string.
2. this func loops around the bottom of the current buffer
and continues searching at the top.
*/
int text_document::find_next(int start_iter=cur_doc->cur_line,
    int start_pos=cur_doc->cur_col) {
    int temp_buf;
    int choice;
    if(search_string.empty()) {
        printmsg("No current search string.");
        return(-1);
    }
    if(start_iter==cur_line && start_pos==cur_col)

return(find_string(search_string,cur_line,cur_col+search_string.size())
);

```

```

else
    return(find_string(search_string,cur_line,cur_col));
}

/* find_prev - func to find the previous occurrence (if any)
           of the previously set 'search_string' variable.
params: start position (line,col) pair.
return: -1 if no search string, 1 if found, 0 if not found.

note: 1. this func can be used only after 'find_string' has located
       an occurrence of the search string.
       2. this func loops around the top of the current buffer
       and continues searching at the bottom.
*/

int text_document::find_prev(int start_iter=cur_doc->cur_line,
    int start_pos=cur_doc->cur_col) {
    int choice,end_iter;
    if(search_string.empty()) {
        printmsg("No current search string.");
        return(-1);
    }
    end_iter=start_iter;
    search_line=start_iter; search_pos=start_pos;
    if(search_pos<=0)
        search_pos=0;
    if((choice=text_buf[search_line]->
substr(0,search_pos).rfind(search_string))==string::npos) {
        --search_line; search_pos=text_buf[search_line]->size();
        while(search_line!=end_iter) {
            if(search_line==--1) {
                printmsg("Search wrapped around TOP of text, continuing at
BOTTOM.");
                search_line=num_lines();
            }
            else if((choice=text_buf[search_line]-
>rfind(search_string))!=string::npos)
                break;
            search_line--;
        }
        if(choice==string::npos)
            choice=text_buf[search_line]->rfind(search_string);
    }
    if(choice==string::npos) {
        search_string="";
        search_line=search_pos=-1;
        return(0);
    }
    search_pos=choice;
    cur_line=search_line;
    gotoline(search_line+1);
    cur_col=(int)search_pos; ypos=line_ypos(); xpos=tab_col();
    wmove(text_win,ypos+xpos/width,xpos%width);
}

```

```

show_text();
getyx(text_win,ypos,xpos);
wchgat(text_win,search_string.size(),A_REVERSE,COLOR_PAIR(1),NULL);
if(xpos+search_string.size()>width) {
    mvwchgat(text_win,ypos+1,0,search_string.size()-(width-xpos),
A_REVERSE,COLOR_PAIR(1),NULL);
    wmove(text_win,ypos,xpos);
}
return(1);
}

/* find_and_replace - func to find a specified string
and replace it with another.
params: find_str - string to find.
        search_str - string to replace it with.
        confirm - switch to indicate replace confirmation.
return: count of number occurrences replaced.
*/
int text_document::find_and_replace(string find_str,string rep_str,bool
confirm) {
    int choice=0,i=-1;
    top_line=cur_line=0; first_char=cur_col=0;
    if(!find_string(find_str))
        return(i);
    search_string=find_str;
    for(i=0,choice=0;choice!=27;) {
        wnoutrefresh(text_win);
        if(confirm) {
            printmsg("Replace this occurrence (Yes/No/All/Cancel) ?: ");
            doupdate();
            switch(choice=getch()) {
                case 'a': case 'A': confirm=FALSE;
                case 'Y': case 'y': text_buf[search_line]->
replace(search_pos,search_string.size(),rep_str);
                is_modified=TRUE; i++; break;
                case 'n': case 'N': break;
                default: case 'c': case 'C': choice=27; break;
            }
        }
        else {
            text_buf[search_line]->replace(search_pos,find_str.size(),rep_str);
            is_modified=TRUE, i++;
        }
        if(choice!=27 && !
find_string(search_string,cur_line,cur_col+rep_str.size(),FALSE))
            break;
    }
    show_text(); doupdate();
    search_string=""; search_line=search_pos=-1; return(i);
}

```

```

/* copy_text - func to copy selected text block to the clipboard.
params: the clipboard buffer.
return: true if copy succeeded, else false.
*/
bool text_document::copy_text() {
    int ypos,xpos,line_y=0,start,stop,temp;
    getyx(text_win,ypos,xpos);
    if(beg_line== -1) {
        printmsg("Begin marker not yet set.");
        return(false);
    }
    if(cur_line < beg_line || (cur_line == beg_line && beg_col > cur_col)) {
        temp = beg_line; beg_line = cur_line; cur_line = temp;
        temp = beg_col; beg_col = cur_col; cur_col = temp;
    }
    if(top_line > beg_line)
        start = 0;
    else if(top_line == beg_line) {
        if(first_char >= tab_col(beg_col))
            start = 0;
        else
            start = tab_col(beg_col) - first_char, line_y = line_ypos(top_line);
    }
    else
        start = tab_col(beg_col), line_y = line_ypos(beg_line);

    if(top_line == cur_line)
        stop = tab_col(cur_col) - first_char;
    else
        stop = tab_col(cur_col);
    show_text();
    if((beg_line == cur_line || top_line == cur_line) && stop - start <= width)
        mvwchgat(text_win, line_y + start / width, start % width, stop - start,
A_REVERSE, COLOR_PAIR(1), NULL);
    else {
        mvwchgat(text_win, line_y + start / width, start % width, -1,
A_REVERSE, COLOR_PAIR(1), NULL);
        for(int i = line_y + start / width + 1; i < line_ypos(cur_line) + stop / width; i++)
            mvwchgat(text_win, i, 0, -1, A_REVERSE, COLOR_PAIR(1), NULL);
        mvwchgat(text_win, line_ypos(cur_line) + stop / width, 0, stop % width,
A_REVERSE, COLOR_PAIR(1), NULL);
    }
    wmove(text_win, ypos, xpos);
    wnoutrefresh(text_win);

    clipboard.empty_buf();
    if(beg_line == cur_line)
        clipboard.text_buf.push_back(new_line(text_buf[beg_line] ->
substr(beg_col, cur_col -
beg_col)));
}

```

```

    else {
        clipboard.text_buf.push_back(new_line(text_buf[beg_line]-
>substr(beg_col)));
        for(int i=beg_line+1;i!=cur_line;i++)
            clipboard.text_buf.push_back(new_line(get_line(i)));
        clipboard.text_buf.push_back(new_line(text_buf[cur_line]-
>substr(0,cur_col)));
    }
    return(true);
}

/* paste_text - func to paste text from clipboard onto current buffer,
   at the current cursor position.
   params: the clipboard buffer.
   return: nothing.
*/
void text_document::paste_text() {
    if(clipboard.is_empty()) {
        printmsg("Clipboard is empty.");
        return;
    }
    insert_text(*clipboard.text_buf[0]);
    cur_col+=clipboard.text_buf[0]->size();
    if(clipboard.num_lines(>1) {
        string temp_str=get_line().substr(cur_col);
        delete_text(text_buf[cur_line]->size()-cur_col);
        cur_line++;
        for(int i=1;i<clipboard.text_buf.size();i++,cur_line++)
            insert_line(cur_line,*clipboard.text_buf[i]);
        cur_line--;
        cur_col=text_buf[cur_line]->size();
        insert_text(temp_str);
    }
    printmsg("Clipboard contents pasted into buffer.");
    is_modified=TRUE;
    int new_col=cur_col;
    gotoline();
    cur_col=new_col;
    new_col=tab_col(cur_col);
    getyx(text_win,ypos,xpos);
    wmove(text_win,ypos+new_col/width,new_col%width);
    show_text();
}

/* cut_text - deletes marked portion of text buffer and puts it onto
the
   clipboard.
   params: nothing.
   return: nothing.
*/

```

```

void text_document::cut_text() {
    if(!copy_text())
        return;
    printmsg("Selected text cut to Clipboard.");
    if(beg_line==cur_line)
        delete_text(cur_col-beg_col,beg_line,beg_col);
    else {
        if(beg_col) {
            delete_text(text_buf[beg_line]->size()-beg_col,beg_line,beg_col);
            beg_line++;
        }
        for(;beg_line!=cur_line;cur_line--)
            delete_line(beg_line);
        if(cur_col==get_line().size())
            delete_line(cur_line);
        else
            delete_text(cur_col,cur_line,0);
    }
    beg_line=-1;
    cur_col=0;
    is_modified=TRUE;
    gotoline();
}

/* set_left_mar - func to set the left margin of current text window
params: mar - margin value
return: nothing
*/
void text_document::set_left_mar(int mar) {
    if(mar<0 || mar>15) {
        msgbox("Out of Bounds","Value should be between 0 and 15.");
        return;
    }
    getmaxyx(bor_win,ypos,xpos);
    left_mar=mar;
    width=xpos-(left_mar+right_mar)-line_num-2;
    wresize(text_win,height,width);
    mvwin(text_win,top_mar+2,line_num+left_mar+1);
    touchwin(text_win);
    touchwin(bor_win); wnoutrefresh(bor_win);
    first_char=cur_col=0;
    cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
}

/* set_right_mar - func to set the right margin of current text window.
params: mar - margin value.
return: nothing
*/

```

```

void text_document::set_right_mar(int mar) {
    if(mar<0 || mar>15) {
        msgbox("Out of Bounds","Value should be between 0 and 15.");
        return;
    }
    right_mar=mar;
    getmaxyx(bor_win,ypos,xpos);
    width=xpos-(left_mar+right_mar)-line_num-2;
    wresize(text_win,height,width);
    touchwin(text_win);
    touchwin(bor_win); wnoutrefresh(bor_win);
    first_char=cur_col=0;
    cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
}

/* set_beg_mar - func to set the begin margin of current text window.
params: mar - margin value.
return: nothing
*/
void text_document::set_beg_mar(int mar) {
    if(mar<0 || mar>5) {
        msgbox("Out of Bounds","Value should be between 0 and 5.");
        return;
    }
    top_mar=mar;
    getmaxyx(bor_win,ypos,xpos);
    height=ypos-(top_mar+bot_mar)-2; max_y=height-1;
    wresize(text_win,height,width);
    mvwin(text_win,top_mar+2,line_num+left_mar+1);
    touchwin(text_win);
    touchwin(bor_win); wnoutrefresh(bor_win);
    first_char=cur_col=0;
    cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
}

/* set_end_mar - func to set the end margin of current text window.
params: mar - margin value.
return: nothing
*/
void text_document::set_end_mar(int mar) {
    if(mar<0 || mar>5) {
        msgbox("Out of Bounds"," Value should be between 0 and 5.");
        return;
    }
    bot_mar=mar;
    getmaxyx(bor_win,ypos,xpos);
    height=ypos-(top_mar+bot_mar)-2; max_y=height-1;
    wresize(text_win,height,width);
    touchwin(text_win);
    touchwin(bor_win); wnoutrefresh(bor_win);
}

```

```

    first_char=cur_col=0;
    cur_line=top_line;
    show_text();
    wmove(text_win,0,0);
}

/* show_line_num - func to show line numbers.
params: nothing
return: nothing
*/
void text_document::show_line_num() {
    for(line_num=1;(num_lines()/((int)pow(10,line_num))!=0;line_num++);
        line_num+=1;
        set_left_mar(left_mar);
}

/* print_status - prints info about current buffer at bottom line of
screen.
params: nothing
return: nothing
*/
void text_document::print_status() {
    getyx(text_win,ypos,xpos);
    if(is_modified) {
        if(file_id)
            printmsg("Buffer: \"Untitled%d\" (%d of %d open files)\
[%d
line(s)][modified].",file_index(),doc_index+1,doc_vec.size(),num_lines(
));
        else
            printmsg("Buffer: \"%s\" (%d of %d open files) [%d
line(s)][modified].",
file_path.c_str(),doc_index+1,doc_vec.size(),num_lines());
    }
    else {
        if(file_id)
            printmsg("Buffer: \"Untitled%d\" (%d of %d open files)\
[%d line(s)].",file_id,doc_index+1,doc_vec.size(),num_lines());
        else
            printmsg("Buffer: \"%s\" (%d of %d open files) [%d line(s)].",
file_path.c_str(),doc_index+1,doc_vec.size(),num_lines());
    }
    wmove(text_win,ypos,xpos);
}

/* switch_doc - func to show the document pointed to by cur_doc.
params: nothing
return: nothing
note: this func is used in switching between multiple documents.
*/

```

```
void text_document::switch_doc() {  
    cur_doc=doc_vec[doc_index];  
    show_text();  
    touchwin(bor_win); touchwin(text_win);  
    wnoutrefresh(bor_win); wnoutrefresh(text_win);  
    doupdate();  
}
```

End of File #1: step_doc.h

File #2: step_menu.h

```

#include "step_doc.h"

class menu_obj {
    ITEM** menu_items;           /* array of item ptrs */
    MENU* menu_ptr;             /* menu pointer */
    WINDOW *menu_win, *menu_sub, *menu_shadow; /* menu windows */
    string menu_name;           /* menu name string */
    int item_count, menu_id, ypos, xpos;
        /* count of number of item, menu id and display coordinates */

public:
    void refresh();
    menu_obj(const string, const int, char**, char**);
    ~menu_obj();
    void next_item();
    void prev_item();
    int cur_item();
    void set_cur_item(int);
    void set_selectable(const int, bool);
    bool is_selectable(const int);
    int process_item();
    int mouse_click();
    void show(const int, const int);
    void hide();
};

/* menu name string arrays */

char* item_names[][8]={
    {"New", "Open", "Close", "Save", "save As", "Delete", "Insert", "Exit"},
    {"Begin mark", "Cut", "cOpy", "Paste"},
    {"Line numbers", "Go to line", "Beeps", "Page setup", "Color
setup", "Refresh"},
    {"Find string", "find Next", "find Previous", "find and Replace"},
    {"Help", "About"},
    {"Left margin", "Right margin", "Begin margin", "End margin"},
    {"STEP Default", "White-Black", "Green-Black",
    "Yellow-Blue", "Red-Green", "Magenta-Cyan"},
    {"OK", "Cancel"}};

/* menu description string arrays */

char* item_descs[][8]={
    {"<F2>", "<F3>", "<F4>", "<F5>", "<F6>", "<F7>", "<F8>", "<F12>"},
    {"Ctrl-B", "Ctrl-X", "Ctrl-V", "Ctrl-P"},
    {"Ctrl-L", "Ctrl-G", "On/Off", "---->", "---->", 0},
    {"Ctrl-F", "<F9>", "<F10>", "Ctrl-R"},
    {"<F1>", "Ctrl-A"}};

```

```

/* menu object declarations */

menu_obj file_menu("file",8,item_names[0],item_descs[0]);
menu_obj edit_menu("edit",4,item_names[1],item_descs[1]);
menu_obj options_menu("options",6,item_names[2],item_descs[2]);
menu_obj tools_menu("tools",4,item_names[3],item_descs[3]);
menu_obj help_menu("help",2,item_names[4],item_descs[4]);
menu_obj marg_menu("margin",4,item_names[5]);
menu_obj colors_menu("colors",6,item_names[6]);
menu_obj ok_menu("ok",2,item_names[7]);

/* the menu_obj class member functions */

/* menu_obj class constructor.
params: menu_name - menu_name string.
       num_items - number of items in menu.
       item_names - array of item name strings.
       item_descs - array of item description string.
*/
menu_obj::menu_obj(const string name,const int num_items,
                  char** item_names,char**
item_descs=NULL) {
    static int cur_id=0;
    menu_id=cur_id++;
    menu_items=new ITEM*[num_items];
    item_count=num_items;
    for(int i=0;i<num_items;i++) {
        if(item_descs)
            menu_items[i]=new_item(item_names[i],item_descs[i]);
        else
            menu_items[i]=new_item(item_names[i],0);
    }
    menu_ptr=new_menu(menu_items);
    menu_name=name;
    set_menu_mark(menu_ptr,"");
    set_menu_fore(menu_ptr,A_REVERSE|A_BOLD|COLOR_PAIR(5));
    set_menu_back(menu_ptr,A_NORMAL|COLOR_PAIR(2));
    set_menu_grey(menu_ptr,A_REVERSE|A_BOLD|COLOR_PAIR(3));
    menu_opts_off(menu_ptr,O_NONCYCLIC);
}

/* menu_obj class destructor.
    frees up storage for menu items and the menu itself.
*/
menu_obj::~menu_obj() {
    free_menu(menu_ptr);
    for(int i=0;i<item_count;i++)
        free_item(menu_items[i]);
    delete [] menu_items;
}

```

```

/* show - func to show the current menu in its associated window pair.
params: (ypos,xpos) coordinate pair indicating position
        at which to show the menu.
return: nothing.
*/
void menu_obj::show(const int y,const int x) {
    int rows,cols;
    ypos=y; xpos=x;
    mvchgat(y-1,x,menu_name.size()+2,A_REVERSE|A_BOLD|COLOR_PAIR(2),
COLOR_PAIR(2),NULL);
    touchline(stdscr,0,1); wnoutrefresh(stdscr);
    scale_menu(menu_ptr,&rows,&cols);
    menu_shadow=newwin(rows+2,cols+2,y+1,x+1);
    menu_win=newwin(rows+2,cols+2,ypos,x);
    menu_sub=newwin(rows,cols,y+1,x+1);
    set_menu_win(menu_ptr,menu_win); set_menu_sub(menu_ptr,menu_sub);
    wbkgd(menu_shadow,' '|A_REVERSE|COLOR_PAIR(2));
    wbkgd(menu_win,' '|COLOR_PAIR(2));
    keypad(menu_win,TRUE); box(menu_win,0,0); nodelay(menu_win,TRUE);
    wbkgd(menu_sub,' '|COLOR_PAIR(2)); keypad(menu_sub,TRUE);
    touchwin(menu_shadow); wnoutrefresh(menu_shadow);
    touchwin(menu_win); wnoutrefresh(menu_win);
    touchwin(menu_sub); wnoutrefresh(menu_sub); doupdate();
    post_menu(menu_ptr);
    set_cur_item(0);
}

/* hide - func to close the current menu.
params: nothing.
return: nothing.
*/
void menu_obj::hide() {
    if(unpost_menu(menu_ptr)!=E_OK)
        return;
    mvchgat(ypos-1,xpos,menu_name.size()+2,A_NORMAL|COLOR_PAIR(4),
COLOR_PAIR(4),NULL);
    mvchgat(ypos-1,xpos+1,1,A_NORMAL|COLOR_PAIR(6),COLOR_PAIR(4),NULL);
    touchline(stdscr,0,1); wnoutrefresh(stdscr);
    delwin(menu_sub);
    delwin(menu_win);
    delwin(menu_shadow);
    if(cur_doc) {
        touchwin(cur_doc->bor_win); touchwin(cur_doc->text_win);
        wnoutrefresh(cur_doc->bor_win); wnoutrefresh(cur_doc->text_win);
    }
}

```

```
/* refresh - refreshes the menu windows.
params: nothing.
return: nothing.
*/
void menu_obj::refresh() {
    touchwin(menu_shadow); wnoutrefresh(menu_shadow);
    touchwin(menu_win); wnoutrefresh(menu_win);
    touchwin(menu_sub); wnoutrefresh(menu_sub); douupdate();
}

/* next_item - func to select the next item in the current menu.
params: nothing.
return: nothing.
*/
void menu_obj::next_item() {
    menu_driver(menu_ptr,REQ_NEXT_ITEM);
}

/* prev_item - func to select the previous item in the current menu.
params: nothing.
return: nothing.
*/
void menu_obj::prev_item() {
    menu_driver(menu_ptr,REQ_PREV_ITEM);
}

/* cur_item - func to return the current item index in the current
menu.
params: nothing.
return: the (zero index) current item index.
*/
int menu_obj::cur_item() {
    return(item_index(current_item(menu_ptr)));
}

/* set_cur_item - sets the current item in the current menu.
params: item index to set as current item.
return: nothing.
*/
void menu_obj::set_cur_item(int item_no) {
    set_current_item(menu_ptr,menu_items[item_no]);
}

/* set_selectable - func to set the selectablility
of a menu item in the current menu.
params: item_num - index of item whose selectablility is to be
set/unset.
opt - switch to indicate whether to set(1) or unset(0)
the item's selectability.
return: nothing.
*/
```

```

void menu_obj::set_selectable(const int item_num,bool opt) {
    if(opt)
        item_opts_on(menu_items[item_num],O_SELECTABLE);
    else
        item_opts_off(menu_items[item_num],O_SELECTABLE);
}

/* is_selectable - func to determine selectablility
   of a menu item in the current menu.
params: item_num - index of item whose selectability is to be
determined.
return: true if item is selectable, false otherwise.
*/
bool menu_obj::is_selectable(const int item_num) {
    return((item_opts(menu_items[item_num])&O_SELECTABLE)?1:0);
}

/* mouse_click - mouse click processing for the current menu.
params: nothing.
return: flag indicating the coordinates and button state of the click.
*/
int menu_obj::mouse_click() {
    MEVENT mouse_event;
    getmouse(&mouse_event);
    if(wmouse_trafo(menu_sub,&mouse_event.y,&mouse_event.x,FALSE)) {
        if(mouse_event.bstate&BUTTON1_PRESSED) {
            set_cur_item(mouse_event.y);
            menu_driver(menu_ptr,KEY_MOUSE);
            return(1);
        }
        else if(mouse_event.bstate&BUTTON1_RELEASED)
            return(0);
    }
    else
        return(-1);
}

/* process_item - func for doing navigation and
                  selection processing for the menus.
params: nothing.
return: nothing.
note: this func contains processing code for all the menus.
      common features of the menus are grouped together.
*/
int menu_obj::process_item() {
    int key=0,choice=0;
    while(key!=27) {
        refresh();
        switch(key=wgetch(menu_sub)) {
            case KEY_DOWN: next_item(); break;
            case KEY_UP: prev_item(); break;
            case KEY_MOUSE:
                choice=mouse_click();
                if(!choice)

```

```
        ungetch('\n');
    else if(choice== -1)
        key=27;
    break;

case 28 ... 30:
case 32 ... 153:
    switch(menu_id) {
        case 0: switch(key) {
            case 'n':case 'N': set_cur_item(0); break;
            case 'o':case 'O': set_cur_item(1); break;
            case 'c':case 'C': set_cur_item(2); break;
            case 's':case 'S': set_cur_item(3); break;
            case 'a':case 'A': set_cur_item(4); break;
            case 'd':case 'D': set_cur_item(5); break;
            case 'i':case 'I': set_cur_item(6); break;
            case 'e':case 'E': set_cur_item(7); break;
            default: continue;
        }
        ungetch('\n'); break;
        case 1: switch(key) {
            case 'b':case 'B': set_cur_item(0); break;
            case 'c':case 'C': set_cur_item(1); break;
            case 'o':case 'O': set_cur_item(2); break;
            case 'p':case 'P': set_cur_item(3); break;
            default: continue;
        }
        ungetch('\n'); break;
        case 2: switch(key) {
            case 'l':case 'L': set_cur_item(0); break;
            case 'g':case 'G': set_cur_item(1); break;
            case 'b':case 'B': set_cur_item(2); break;
            case 'p':case 'P': set_cur_item(3); break;
            case 'c':case 'C': set_cur_item(4); break;
            case 'r':case 'R': set_cur_item(5); break;
            default: continue;
        }
        ungetch('\n'); break;
        case 3: switch(key) {
            case 'f':case 'F': set_cur_item(0); break;
            case 'n':case 'N': set_cur_item(1); break;
            case 'p':case 'P': set_cur_item(2); break;
            case 'r':case 'R': set_cur_item(3); break;
            default: continue;
        }
        ungetch('\n'); break;
        case 4: switch(key) {
            case 'h':case 'H': set_cur_item(0); break;
            case 'a':case 'A': set_cur_item(1); break;
            default: continue;
        }
        ungetch('\n'); break;
    }
    break;
```

```

case '\n':
    if(!is_selectable(cur_item()))
        break;
    switch(menu_id) {
    case 0: switch(cur_item()) {
        case 0: ungetch(KEY_F0+2); break;
        case 1: ungetch(KEY_F0+3); break;
        case 2: ungetch(KEY_F0+4); break;
        case 3: ungetch(KEY_F0+5); break;
        case 4: ungetch(KEY_F0+6); break;
        case 5: ungetch(KEY_F0+7); break;
        case 6: ungetch(KEY_F0+8); break;
        case 7: ungetch(KEY_F0+12); break;
    }
        break;
    case 1: switch(cur_item()) {
        case 0: ungetch(2); break;
        case 1: ungetch(24); break;
        case 2: ungetch(22); break;
        case 3: ungetch(16); break;
    }
        break;
    case 2: switch(cur_item()) {
        case 0: ungetch(12); break;
        case 1: ungetch(7); break;
        case 2: if(beeps_on)
            beeps_on=FALSE, printmsg("Beeps turned OFF.");
            else
            beeps_on=TRUE, printmsg("Beeps turned ON.");
            break;
        case 3: marg_menu.show(4,36);
            if(marg_menu.process_item()==1)
            { marg_menu.hide(); continue; }
            marg_menu.hide(); break;
        case 4: colors_menu.show(5,36);
            if(colors_menu.process_item()==1)
            { colors_menu.hide(); continue; }
            colors_menu.hide(); break;
        case 5: cur_doc->show_text(); break;
    }
        break;
    case 3: switch(cur_item()) {
        case 0: ungetch(6); break;
        case 1: ungetch(KEY_F0+9); break;
        case 2: ungetch(KEY_F0+10); break;
        case 3: ungetch(18); break;
    }
        break;
    case 4: switch(cur_item()) {
        case 0: ungetch(KEY_F0+1); break;
        case 1: ungetch(1); break;
    }
        break;

```

```

case 5:
    switch(cur_item()) {
    case 0:
        if(!inputbox("Enter Left Margin: ", "%d", &choice))
            break;
        cur_doc->set_left_mar(choice);
        break;
    case 1:
        if(!inputbox("Enter Right Margin: ", "%d", &choice))
            break;
        cur_doc->set_right_mar(choice);
        break;
    case 2:
        if(!inputbox("Enter Begin Margin: ", "%d", &choice))
            break;
        cur_doc->set_beg_mar(choice);
        break;
    case 3:
        if(!inputbox("Enter End Margin: ", "%d", &choice))
            break;
        cur_doc->set_end_mar(choice);
        break;
    }
    break;

case 6:
    choice=0;
    switch(cur_item()) {
    case 0: init_pair(1,COLOR_WHITE,COLOR_BLUE); break;
    case 1: init_pair(1,COLOR_WHITE,COLOR_BLACK); break;
    case 2: init_pair(1,COLOR_GREEN,COLOR_BLACK); break;
    case 3: init_pair(1,COLOR_YELLOW,COLOR_BLUE);
            choice=A_BOLD; break;
    case 4: init_pair(1,COLOR_RED,COLOR_GREEN); break;
    case 5: init_pair(1,COLOR_MAGENTA,COLOR_CYAN); break;
    }
    for(int i=0;i<doc_vec.size();i++)
        wbgd(doc_vec[i]->text_win, ' ' | COLOR_PAIR(1) | choice);
    break;

case 7: return(cur_item()?0:1);
}
key=27; break;

case KEY_RIGHT:
    switch(menu_id) {
    case 0: ungetch('E'); break;
    case 1: ungetch('O'); break;
    case 2: if(cur_item()==3) {
            marg_menu.show(4,36);
            if(marg_menu.process_item()==1)
                { marg_menu.hide(); continue; }
            marg_menu.hide();
        }
    }

```

```
        else if(cur_item()==4) {
            colors_menu.show(5,36);
            if(colors_menu.process_item()==1)
                { colors_menu.hide(); continue; }
            colors_menu.hide();
        }
        else
            ungetch('T');
        break;
    case 3: ungetch('H'); break;
    case 4: ungetch('F'); break;
    case 5: case 6: continue;
}
ungetch(27); key=27; break;
case KEY_LEFT:
    switch(menu_id) {
        case 0: ungetch('H'); break;
        case 1: ungetch('F'); break;
        case 2: ungetch('E'); break;
        case 3: ungetch('O'); break;
        case 4: ungetch('T'); break;
        case 5: return(1);
        case 6: return(1);
    }
    ungetch(27); key=27; break;
case 27: if((choice=wgetch(menu_win))!=ERR)
        ungetch(choice), ungetch(27);
default: key=27; break;
}
}
return(0);
}
```

End of File #2: step_menu.h

File #3: step_main.cpp

```

#include "step_menu.h"

/* process_signal func - does appropriate processing depending signal
type.
params: signal number
return: nothing
*/
void process_signal(int sig_num) {
    static time_t cur_time;
    static int y,x;

    switch(sig_num) {

        case 0: endwin(); system("clear");
        printf("Memory allocation error. STEP terminated abnormally.\n");
        break;

        case SIGINT: case SIGABRT:
        endwin(); system("clear");
        printf("STEP terminated abnormally.\n"); break;

        case SIGSEGV:
        endwin(); system("clear");
        printf("STEP terminated due to a segmentation fault.\n");
        printf("If you have found a bug in STEP, report it to:\n");
        printf("\nsrivasnchennu@yahoo.com\n");
        printf("\nwith a complete description.\n");
        break;

        case SIGWINCH:
        endwin(); system("clear"); break;

        case SIGALRM: getsyx(y,x);
            alarm(1); time(&cur_time);
            mvprintw(0, COLS-26, "%s", ctime(&cur_time));
        wnoutrefresh(stdscr);
            setsyx(y,x); douupdate();
            return;
    }

    for(doc_index=0;doc_index!=doc_vec.size();doc_index++)
        delete(doc_vec[doc_index]);
    doc_vec.clear();
    alarm(0); exit(-1);
}

```

```

/* new_doc - creates a new text_document object, and does error
processing.
params: optional filename to load the new object's buffer with.
return: pointer to the newly created object.
*/
text_document* new_doc(const string& filename="") {
    text_document* temp_docp=new text_document(filename);
    if(!temp_docp)
        process_signal(0);
    return(temp_docp);
}

/* print message func - prints a simple message to last line of screen
param: variable arguments
return: nothing
*/
void printmsg(char* format,...) {
    move(LINES-1,1);
    clrtoeol();
    if(ins_mode_on)
        mvaddstr(LINES-1,COLS-7,"INSERT");
    move(LINES-1,1);
    va_list ap;
    va_start(ap,format);
    vwprintw(stdscr,format,ap);
    va_end(ap);
    wnoutrefresh(stdscr);
}

/* msgbox - func to display a message prompt in a window.
params: variable args.
        title_str - prompt string to be displayed as the window title.
return: true if user selected OK, false if user selected CANCEL.
*/
bool msgbox(char* title_str,char* format,...) {
    int choice=-1;
    curs_set(0);
    WINDOW* shadow=newwin(LINES/2+1,COLS/2,LINES/2-LINES/4+1,COLS/2-
COLS/4+1);
    WINDOW* msg_win=newwin(LINES/2+1,COLS/2,LINES/2-LINES/4,COLS/2-
COLS/4);
    WINDOW* msg_sub=newwin(LINES/2-8,COLS/2-4,LINES/2-LINES/4+4,COLS/2-
COLS/4+2);
    wbkgd(shadow,' '|COLOR_PAIR(2)|A_REVERSE);
    wbkgd(msg_win,' '|COLOR_PAIR(2)); keypad(msg_win,TRUE);
    box(msg_win,0,0);
    wbkgd(msg_sub,' '|COLOR_PAIR(2)); keypad(msg_sub,TRUE);

    mvwaddstr(msg_win,1,COLS/4-(strlen(title_str)/2),title_str);
    mvwchgat(msg_win,1,1,COLS/2-
2,A_BOLD|COLOR_PAIR(5),COLOR_PAIR(5),NULL);
    va_list ap;
    va_start(ap,format);
    vwprintw(msg_sub,format,ap);

```

```

va_end(ap);
wnoutrefresh(shadow); wnoutrefresh(msg_win);
wnoutrefresh(msg_sub); doupdate();
ok_menu.show(LINES-LINES/4-6, COLS/2-5);

while(choice== -1) {
    ok_menu.refresh();
    switch(wgetch(msg_sub)) {
        case KEY_MOUSE: if(!ok_menu.mouse_click())
            ungetch('\n');
            break;
        case KEY_UP: ok_menu.prev_item(); break;
        case '\t': case KEY_DOWN: ok_menu.next_item(); break;
        case '\n': choice=(ok_menu.cur_item()?0:1); break;
        case 27: choice=0; break;
    }
}

ok_menu.hide();
delwin(msg_sub); delwin(msg_win); delwin(shadow);
if(cur_doc) {
    touchwin(cur_doc->bor_win); wnoutrefresh(cur_doc->bor_win);
    touchwin(cur_doc->text_win); wnoutrefresh(cur_doc->text_win);
}
doupdate(); curs_set(1);
return((bool)choice);
}

/* inputbox - func to accept data from user, in window.
   params: prompt - prompt string to be displayed in the window.
           fmt - string indicating the datatype of the input data.
           data - pointer to memory into which input data is to be put.
   return: true if user selected OK, false if user selected CANCEL.
*/
bool inputbox(char* prompt, char* fmt, void* data) {
    int key=-1;
    bool choice;
    text_document ip_doc(LINES/2-10, COLS/2-4);

    cur_doc=&ip_doc;
    WINDOW* shadow=newwin(LINES/2+1, COLS/2, LINES/2-LINES/4+1, COLS/2-
COLS/4+1);
    cur_doc->bor_win=newwin(LINES/2+1, COLS/2, LINES/2-LINES/4, COLS/2-
COLS/4);
    cur_doc->text_win=newwin(LINES/2-10, COLS/2-4, LINES/2-LINES/4+5, COLS/2-
COLS/4+2);
    wbkgd(shadow, ' ' | COLOR_PAIR(2) | A_REVERSE);
    wbkgd(cur_doc->bor_win, ' ' | COLOR_PAIR(2)); keypad(cur_doc-
>bor_win, TRUE);
    wbkgd(cur_doc->text_win, ' ' | COLOR_PAIR(2)); keypad(cur_doc-
>text_win, TRUE);

    box(cur_doc->bor_win, 0, 0);
    mvwaddstr(cur_doc->bor_win, 1, COLS/4-5, "STEP Prompt");
}

```

```

mvwchgat(cur_doc->bor_win,1,1,COLS/2-
2,A_BOLD|COLOR_PAIR(5),COLOR_PAIR(5),NULL);
mvwaddstr(cur_doc->bor_win,3,4,prompt);
wnoutrefresh(shadow); wnoutrefresh(cur_doc->bor_win);
wnoutrefresh(cur_doc->text_win); douupdate();
ok_menu.show(LINES-LINES/4-6,COLS/2-5);
wmove(cur_doc->text_win,0,0);

while(key!='\n') {
    ok_menu.refresh();
    cur_doc->show_text(); wnoutrefresh(cur_doc->text_win); douupdate();
    switch(key=wgetch(cur_doc->text_win)) {
        case KEY_MOUSE: if(!ok_menu.mouse_click())
            ungetch('\n');
            break;
        case KEY_UP: ok_menu.prev_item(); break;
        case '\t': case KEY_DOWN: ok_menu.next_item(); break;
        case KEY_RIGHT: cur_doc->move_right(); break;
        case KEY_LEFT: cur_doc->move_left();
        case 28 ... 30: case 32 ... 153:
            if(cur_doc->get_line().size()>=cur_doc->width*2-1) {
                if(beeps_on) beep();
                break;
            }
            if(!ins_mode_on)
                cur_doc->delete_text();
            cur_doc->insert_text((char *)&key);
            cur_doc->move_right();
            break;
        case KEY_DC:
            cur_doc->delete_text(); break;
        case KEY_BACKSPACE: cur_doc->move_left();
            cur_doc->delete_text(); break;
        case 27: choice=false; key='\n'; break;
        case '\n': if(!ok_menu.cur_item()) {
            choice=true;
            switch(*(++fmt)) {
                case 'd':
                    *((int *)data)=strtol(cur_doc->
>get_line().c_str(),NULL,10);
                    break;
                case 's': *((string *)data)=cur_doc->get_line(); break;
            }
        }
        else
            choice=false;
        break;
    }
}
delwin(shadow);
ok_menu.hide();
doc_vec[doc_index]->switch_doc();
return(choice);
}

```

```

/* textbox - func to display specified text in a window
   of specified size.
params: title_str - prompt string to be displayed as title of window
       text - text to be displayed in the window.
       height - height of text window.
       width - width of text window.
return: nothing.
*/
void textbox(char* title_str,char* text,int height,int width) {
    curs_set(0);
    WINDOW* text_win=newwin(height,width,LINES/2-height/2,COLS/2-width/2);
    WINDOW* text_sub=newwin(height-3,width-4,LINES/2-height/2+2,COLS/2-
width/2+2);
    WINDOW* shadow=newwin(height,width,LINES/2-height/2+1,COLS/2-
width/2+1);
    wbkgd(shadow,' '|COLOR_PAIR(2)|A_REVERSE);
    wbkgd(text_win,' '|COLOR_PAIR(2)); keypad(text_win,TRUE);
    wbkgd(text_sub,' '|COLOR_PAIR(2)); keypad(text_sub,TRUE);
    box(text_win,0,0);
    mvwaddstr(text_win,1,width/2-strlen(title_str)/2-2,title_str);
    mvwchgat(text_win,1,1,width-
2,A_BOLD|COLOR_PAIR(5),COLOR_PAIR(5),NULL);
    mvwprintw(text_sub,0,0,text);
    wnoutrefresh(shadow); wnoutrefresh(text_win);
    wnoutrefresh(text_sub); douupdate();
    wgetch(text_win);
    delwin(shadow); delwin(text_sub); delwin(text_win);
    if(cur_doc) {
        touchwin(cur_doc->bor_win); wnoutrefresh(cur_doc->bor_win);
        touchwin(cur_doc->text_win); wnoutrefresh(cur_doc->text_win);
    }
    douupdate(); curs_set(1);
}

/* process_input - the main input processing function.
   does conditional switching depending on input character.
params: nothing.
return: 0 on successful exit, otherwise -1.
*/
int process_input() {
    int ypos,xpos,choice,key;
    string filename,find_str,rep_str;
    MEVENT mouse_event;

    while(1) {
        mvwhline(cur_doc->bor_win,LINES-
3,1,ACS_HLINE|COLOR_PAIR(2)|A_REVERSE,COLS-2);
        wattrset(cur_doc->bor_win,A_REVERSE);
        wprintw(cur_doc->bor_win," %d:%d ",cur_doc->cur_line+1,cur_doc-
>cur_col+1);
        wnoutrefresh(cur_doc->bor_win);
        wnoutrefresh(cur_doc->text_win); douupdate();
        key=wgetch(cur_doc->text_win);

```

```

    getyx(cur_doc->text_win,ypos,xpos);
    printmsg("F1 Help  F2 New  F3 Open  F4 Close  F5 Save  F6 Save as
F12 Exit");

/* the main input processing switch statement */

    switch(key) {

/* mouse button press processing.  activates appropriate menu. */
    case KEY_MOUSE:
        getmouse(&mouse_event);
        if(mouse_event.bstate&BUTTON1_RELEASED && mouse_event.y == 0) {
            switch(mouse_event.x) {
                case 1 ... 4: ungetch('F'); ungetch(27); break;
                case 8 ... 11: ungetch('E'); ungetch(27); break;
                case 15 ... 21: ungetch('O'); ungetch(27); break;
                case 25 ... 29: ungetch('T'); ungetch(27); break;
                case 33 ... 36: ungetch('H'); ungetch(27); break;
            }
        }
        break;

/* ALT key commands */
    case 27:
        if((choice=wgetch(cur_doc->bor_win))!=ERR)
            ungetch(choice);
        else
            break;
        curs_set(0);
        key=wgetch(cur_doc->text_win);
        switch(key) {

/* menu command processing */
/* file menu */
        case 'F': case 'f':
            file_menu.show(1,1); file_menu.process_item();
            file_menu.hide(); break;

/* edit menu */
        case 'E': case 'e':
            edit_menu.show(1,8); edit_menu.process_item();
            edit_menu.hide(); break;

/* options menu */
        case 'O': case 'o':
            options_menu.show(1,15); options_menu.process_item();
            options_menu.hide(); break;

/* tools menu */
        case 'T': case 't':
            tools_menu.show(1,25); tools_menu.process_item();
            tools_menu.hide(); break;

```

```

/* help menu */
case 'H': case 'h':
help_menu.show(1,33); help_menu.process_item();
help_menu.hide(); break;

/* document switching */
case '\t':
doc_index++;
if(doc_index==doc_vec.size())
doc_index=0;
doc_vec[doc_index]->switch_doc(); cur_doc->print_status();
break;

/* document switching */
case 48 ... 57:
key-=48;
if(key==0) key=10;
if(key>doc_vec.size()) {
printmsg("Error: Out of Range. Current file count:
%d.",doc_vec.size());
break;
}
key--; doc_index=key;
doc_vec[doc_index]->switch_doc(); cur_doc->print_status();
break;
}
curs_set(1);
break;

/* alphanumeric character and tab processing */
case 28 ... 30:
case 32 ... 153:
case '\t':
if(!ins_mode_on)
cur_doc->delete_text();
cur_doc->insert_text((char *)&key);
cur_doc->move_right();
cur_doc->show_text();
cur_doc->is_modified=TRUE;
break;

/* BACKSPACE key processing - supports bkspacing over multiple lines
*/
case KEY_BACKSPACE:

if(!cur_doc->cur_col && !cur_doc->cur_line) {
if(beeps_on) beep();
break;
}
else if(!cur_doc->cur_col) {
cur_doc->move_up();
cur_doc->move_end();
cur_doc->insert_text(

```

```

        cur_doc->get_line(cur_doc->cur_line+1));
    cur_doc->delete_line(cur_doc->cur_line+1);
}
else {
    cur_doc->move_left();
    cur_doc->delete_text();
}
cur_doc->show_text();
cur_doc->is_modified=TRUE;
break;

/* DELETE key processing */
case KEY_DC:
    if(cur_doc->cur_line==cur_doc->num_lines()-1 &&
        cur_doc->cur_col==cur_doc->get_line().size()) {
        if(beeps_on) beep();
        break;
    }
    else if(cur_doc->cur_col==cur_doc->get_line().size()) {
        cur_doc->insert_text(
            cur_doc->get_line(cur_doc->cur_line+1));
        cur_doc->delete_line(cur_doc->cur_line+1);
        if(cur_doc->line_ypos()+cur_doc->tab_col(
            cur_doc->get_line().size())/
            cur_doc->width>cur_doc->max_y) {
            cur_doc->move_up();
            cur_doc->move_down();
            cur_doc->move_home();
        }
    }
    else
        cur_doc->delete_text();
    cur_doc->show_text();
    cur_doc->is_modified=TRUE;
    break;

/* delete line processing */
case 4:
    getyx(cur_doc->text_win,ypos,xpos);
    if(cur_doc->is_empty()) {
        printmsg("Empty Buffer.");
        if(beeps_on) beep();
        break;
    }
    if(cur_doc->num_lines()==1) {
        cur_doc->delete_line();
        cur_doc->insert_line(0,"");
        cur_doc->cur_line=cur_doc->cur_col=0;
        cur_doc->is_modified=0;
        cur_doc->show_text();
        break;
    }
    if(cur_doc->cur_line==cur_doc->num_lines()-1) {
        cur_doc->move_up();

```

```

    cur_doc->delete_line(cur_doc->cur_line+1);
}
else
    cur_doc->delete_line();
cur_doc->is_modified=TRUE;
cur_doc->cur_col=0;
cur_doc->show_text();
wmove(cur_doc->text_win,cur_doc->line_ypos(),0);
break;

/* delete word processing - supports partial word delete */
case 23:
    if(cur_doc->get_line().empty()) {
        if(beeps_on) beep();
        printmsg("Cannot delete words in empty line.");
        break;
    }
    if(cur_doc->cur_col==cur_doc->get_line().size()) {
        beep();
        break;
    }
    choice=cur_doc->get_line().find_first_of(' ',cur_doc->cur_col);
    if(choice==cur_doc->cur_col)
        cur_doc->delete_text();
    else {
        if(choice==string::npos)
            choice=cur_doc->get_line().size();
        for(int i=0;i<choice-cur_doc->cur_col;i++)
            cur_doc->delete_text();
    }
    cur_doc->show_text();
    cur_doc->is_modified=TRUE;
    break;

/* INSERT key processing */
case KEY_IC:
    ins_mode_on=(ins_mode_on?FALSE:TRUE);
    printmsg("");
    break;

/* ENTER key processing */
case '\n':
    cur_doc->insert_line(cur_doc->cur_line+1,
        cur_doc->get_line().substr(cur_doc->cur_col));
    cur_doc->delete_text(cur_doc->get_line().size()-cur_doc->cur_col);
    cur_doc->move_down();
    cur_doc->move_home();
    cur_doc->show_text();
    cur_doc->is_modified=TRUE;
    break;

```

```

/* Ctrl-F - 'Find String' option processing */
case 6:
    if(!inputbox("Enter search string: ", "%s", &find_str))
        break;
    if(!cur_doc->find_string(find_str))
        msgbox("Search Failed", "Search string not found.");
    break;

/* Ctrl-R - 'Replace String' option processing */
case 18:
    if(!inputbox("Enter string to be replaced: ", "%s", &find_str))
        break;
    if(!inputbox("Enter string to replace it with:", "%s", &rep_str))
        break;
    if(msgbox("Find and Replace", "Confirm Replace ?: "))
        choice=cur_doc->find_and_replace(find_str, rep_str, 1);
    else
        choice=cur_doc->find_and_replace(find_str, rep_str, 0);
    if(choice== -1)
        msgbox("Replace Unsuccessful", "Search string not found.");
    else if(!choice)
        printmsg("Replace operation aborted.");
    else
        printmsg("Replaced %d occurrence(s) of \"%s\" with \"%s\".",
choice, find_str.c_str(), rep_str.c_str());
    break;

/* <F9> - 'Find Next' processing */
case KEY_F0+9:
    if(!cur_doc->find_next())
        msgbox("Search Failed", "Search string not found.");
    break;

/* <F10> - 'Find Previous' option processing */
case KEY_F0+10:
    if(!cur_doc->find_prev())
        msgbox("Search Failed", "Search string not found.");
    break;

/* 'Set Begin Mark' option processing */
case 2:
    cur_doc->beg_line=cur_doc->cur_line;
    cur_doc->beg_col=cur_doc->cur_col;
    printmsg("Begin mark set at %d,%d.", cur_doc->cur_line+1, cur_doc->
beg_col+1);
    break;

/* 'Copy text' option processing */
case 22:
    if(cur_doc->copy_text())
        printmsg("Selected text copied to clipboard.");
    break;

```

```
/* 'Cut text' option processing */
case 24:
    cur_doc->cut_text();
    break;

/* 'Paste text option processing */
case 16:
    cur_doc->paste_text();
    break;

/* 'Show Line Numbers' option processing */
case 12:
    cur_doc->show_line_num();
    break;

/* 'Go To Line' option processing */
case 7:
    if(!inputbox("Enter Line No. to jump to: ", "%d", &choice))
        break;
    cur_doc->gotoline(choice);
    break;

/* Left arrow key processing */
case KEY_LEFT:
    cur_doc->move_left();
    break;

/* Right arrow key processing */
case KEY_RIGHT:
    cur_doc->move_right();
    break;

/* Up arrow key processing */
case KEY_UP:
    cur_doc->move_up();
    break;

/* Down arrow key processing */
case KEY_DOWN:
    cur_doc->move_down();
    break;

/* PAGE DOWN key processing */
case KEY_NPAGE:
    cur_doc->page_down();
    break;

/* PAGE UP key processing */
case KEY_PPAGE:
    cur_doc->page_up();
    break;
```

```

/* HOME key processing */
case KEY_HOME:
    cur_doc->move_home();
    break;
/* END key processing */
case KEY_END:
    cur_doc->move_end();
    break;

/* 'Jump to beginning of file' option processing */
case 8:
    cur_doc->gotoline(1); break;

/* 'Jump to end of file' option processing */
case 11:
    cur_doc->gotoline(cur_doc->num_lines()); break;

/* <F1> - STEP command help */
case KEY_F0+1: textbox("STEP Commands", "
Alt-F File  Alt-E Edit  Alt-O Options  Alt-T Tools  Alt-H Help\n
<F1> Help    <F2> New     <F3> Open   <F4> Close  <F5> Save
<F6> Save As <F7> Delete  <F8> Insert <F11> Status <F12> Exit
Ctrl-B Begin Mark          Ctrl-V Copy text
Ctrl-X Cut text            Ctrl-P Paste text
Ctrl-D Delete line         Ctrl-W Delete word
Ctrl-L Show Line Numbers   Ctrl-G Go To Line
Ctrl-H Go to beginning of file Ctrl-K Go to end of file
Ctrl-F Find String         Ctrl-R Replace String
<F9> Find Next Match      <F10> Find Previous Match
Ctrl-H Display this help   Ctrl-A Display info about STEP",LINES-
8, COLS-8);
    break;

/* Ctrl-A - 'About Step' Box */
case 1: textbox("About STEP", "
Srivas's Text Editing Program.
Ver 1.0.
Copyright 2001 Srivas N. Chennu.

Contact: srivasnchennu@yahoo.com",10,40); break;

/* <F2> - 'New File' option processing */
case KEY_F0+2:
    doc_vec.push_back(new_doc());
    doc_index=doc_vec.size()-1;
    printmsg("New File.");
    doc_vec[doc_index]->switch_doc();
    wmove(cur_doc->text_win,0,0);
    break;

```

```

/* <F3> - 'Open File' option processing */
case KEY_F0+3:
    if(!inputbox("Enter filename to open: ", "%s", &filename))
        break;
    if(!access(filename.c_str(), F_OK)) {
        doc_vec.push_back(new_doc(filename));
        doc_index=doc_vec.size()-1;
        doc_vec[doc_index]->switch_doc();
        wmove(cur_doc->text_win, 0, 0);
    }
    else
        msgbox("File Open Error", "Could not open file
\"%s\".", filename.c_str());
        break;

/* <F4> - 'Close File' option processing */
case KEY_F0+4:
    if(doc_vec.size()==1 && cur_doc->file_index()) {
        printmsg("No open files.");
        break;
    }
    if(cur_doc->is_modified) {
        if(msgbox("Close File", "Save current buffer ?: "))
            if(cur_doc->save_file(cur_doc->file_path))
                break;
    }
    delete(doc_vec[doc_index]);
    doc_vec.erase(doc_vec.begin()+doc_index);
    if(doc_index==doc_vec.size())
        doc_index=0;
    if(doc_vec.empty()) {
        doc_vec.push_back(new_doc());
        doc_index=0;
    }
    doc_vec[doc_index]->switch_doc();
    break;

/* <F5> - 'Save File' option processing */
case KEY_F0+5:
    if(cur_doc->file_path.empty()) {
        if(!inputbox("Enter filename to save as: ", "%s", &filename))
            break;
        if(cur_doc->save_file(filename))
            break;
        cur_doc->file_path=filename;
    }
    else if(cur_doc->save_file())
        break;
    cur_doc->is_modified=FALSE;
    break;

```

```

/* <F6> - 'Save As' option processing */
case KEY_F0+6:
    if(!inputbox("Enter filename to save as: ", "%s", &filename))
        break;
    cur_doc->save_file(filename);
    break;

/* <F7> - 'Delete File' option processing */
case KEY_F0+7:
    if(!inputbox("Enter file to delete: ", "%s", &filename))
        break;
    if(remove(filename.c_str()))
        msgbox("File Delete Error", "File could not be deleted!!.");
    else
        msgbox("File Delete", "File \"%s\" deleted.", filename.c_str());
    break;

/* <F8> - Insert file option processing */
case KEY_F0+8:
    if(!inputbox("Enter filename to insert:", "%s", &filename))
        break;
    cur_doc->insert_file(filename);
    break;

/* <F11> - File Status option processing */
case KEY_F0+11:  cur_doc->print_status(); break;

/* <F12> - 'Exit' option processing */
case KEY_F0+12:
    for(doc_index=0; doc_index!=doc_vec.size(); doc_index++) {
        doc_vec[doc_index]->switch_doc();
        if(cur_doc->is_modified) {
            if(cur_doc->file_path.empty()) {
                if(msgbox("Save to File", "Save buffer \"Untitled%d\" ?: ",
                    cur_doc->file_index())) {
                    if(!inputbox("Enter filename to save as: ", "%s", &filename))
                        break;
                    cur_doc->save_file(cur_doc->file_path=filename);
                }
            }
            else if(msgbox("Save to File", "Save buffer \"%s\" ?: ",
                cur_doc->file_path.c_str()))
                cur_doc->save_file(cur_doc->file_path);
        }
        delete(doc_vec[doc_index]);
    }
    doc_vec.clear();
    return(0);
}
}
}

```

```

/* the main func */
int main(int argc, char* argv[]) {

/* interrupt signal handling */
signal(SIGINT, process_signal);
signal(SIGSEGV, process_signal);
signal(SIGABRT, process_signal);
signal(SIGALRM, process_signal);
signal(SIGWINCH, process_signal);

/* init curses */
initscr();
noecho();
cbreak();
keypad(stdscr, TRUE);

/* init colors */
start_color();
init_pair(1, COLOR_WHITE, COLOR_BLUE);
init_pair(2, COLOR_BLACK, COLOR_WHITE);
init_pair(3, COLOR_CYAN, COLOR_BLACK);
init_pair(4, COLOR_WHITE, COLOR_BLACK);
init_pair(5, COLOR_CYAN, COLOR_WHITE);
init_pair(6, COLOR_WHITE, COLOR_RED);
bkgd(' ' | COLOR_PAIR(2));
mvaddstr(0, 2, "File Edit Options Tools Help");
mvchgat(0, 2, 1, A_NORMAL | COLOR_PAIR(6), COLOR_PAIR(6), NULL);
mvchgat(0, 9, 1, A_NORMAL | COLOR_PAIR(6), COLOR_PAIR(6), NULL);
mvchgat(0, 16, 1, A_NORMAL | COLOR_PAIR(6), COLOR_PAIR(6), NULL);
mvchgat(0, 26, 1, A_NORMAL | COLOR_PAIR(6), COLOR_PAIR(6), NULL);
mvchgat(0, 34, 1, A_NORMAL | COLOR_PAIR(6), COLOR_PAIR(6), NULL);

/* welcome message */
printwmsg("Welcome To STEP. Press F1 for Help.");

/* start the time display */
process_signal(SIGALRM);
doupdate();

/* mouse initialization */
if(NCURSES_MOUSE_VERSION)
    mousemask(ALL_MOUSE_EVENTS, NULL), mouseinterval(50);

/* command line argument processing.
   load any files specified at the command line. */
if(argc > 1)
    for(int i=1; i<argc; i++)
        doc_vec.push_back(new_doc(argv[i]));
    else
        doc_vec.push_back(new_doc());
}

```

```
doc_index=0;
doc_vec[doc_index]->switch_doc();
cur_doc->show_text();
wmove(cur_doc->text_win,0,0);
cur_doc->print_status();

/* call the input processing func */
process_input();

/* close curses and return to shell */
endwin();
alarm(0);
system("clear");
return(0);
}
```

End of File #3: step_main.cpp

THE END