```
main(int argc, char *argv[])
{
int i = 0;

/*if (argc != 5)
        {
        printf("Usage Instructions: dialog config_file learn_file\n");
        printf("*** Exiting, goodbye.\n");
        exit(1);
        }*/
if (argc != 2)
        {
        printf("Usage Instructions: d <ini-file>\n");
        printf("*** Exiting, goodbye.\n");
        exit(1);
        }
readini(argv[1]);
formfile = fdoc;
loadStopWords(sdoc);
numPF = loadFormsList(xdoc);
numForms = loadForms(fdoc);
loadData(cfg, lcfg);
Interactive(lcfg);
}

/*********************************************************************
loaddata : This function will read the configuration files and load the
        information into the relevant global arrays.
*********************************************************************/
loadData(char *filenm, char *file2)
{
char buf[256], word[20];
int i, j, k, l;
int numext;
FILE *fp, *f2;

/******************* open configuration file ********************/
fp = fileOpen(filenm, "r");

/************** open learn(extended thesaurus) file ****************/
f2 = fileOpen(file2, "r");
```

```
prompts = columnTerms = rowTerms = NULL;
scoring = thesaurus = indexList = menuList = NULL;

/* read data in the arrays */
numMenu = loadMenuTree(fp, "[MENUTREE]");
numIndex = readArray(fp, "[PROMPTS]", &prompts, 1, NULL, 0, 0);
numColumn = readArray(fp, "[INDEX]", &columnTerms, 1, &indexList, numIndex, 0);
numOrgRow = readArray(fp, "[THESAURUS]", &rowTerms, 1, &thesaurus, numColumn, 0);
numRow = readArray(f2, "[EXT-THESAURUS]", &rowTerms, 1, &thesaurus, numColumn,
numOrgRow);
numScore = readArray(f2, "[SCORING]", NULL, 0, &scoring, numColumn + 1, 0);

fclose(fp);
fclose(f2);
}

/*************************************************************************
readArray : Reads the file and fills the rows and columns of the given arrays
*************************************************************************/
int readArray(FILE *fp, char *head, char ***ch_array, int ccount, int ***int_array, int icount, int
sp)
{
char buf[256];
int i, j, start = 0, wc = 0;
int k, c;
char **tmparray; /*To store the pointers to the words/numbers from the string*/
c = sp;

if (icount != 0)
        tmparray = (char **)malloc((icount + 1) * sizeof(char *));

fseek(fp, 0, 0); /* Go to Top */

while (fgets(buf, 255, fp) != NULL) /* read lines till end of file */
        {
        allTrim(buf);
        j = strlen(buf);
        if (buf[j - 1] == '\n') buf[j - 1] = 0;
        if (start)
                {
                if (strlen(buf) == 0)  /* if blank line, stop reading */
                        break;
                if (icount == 0) /* i.e. no integer array */
```

```
                addWord(ch_array, buf, ++c);
        else    /* read first word string */
                { /* rest are columns of int array */
                wc = readValues(buf, tmparray);
                c++;
                (*int_array) = (int **)realloc(*int_array, c * sizeof(int *));
                (*int_array)[c - 1] = (int *)malloc(icount * sizeof(int));
                if (ccount != 0)
                        addWord(ch_array, tmparray[0], c);
                else
                        (*int_array)[c - 1][0] = atoi(tmparray[0]);
                for (k = 1; k < icount; k++)
                        if (k < wc)
                                (*int_array)[c - 1][k - ccount] = atoi(tmparray[k]);
                        else
                                (*int_array)[c - 1][k - ccount] = 0;
                }
        }
    else
    if (!strcmp(head, buf))
        start = 1;
    }
return c;
}

/*******************************************************************
loadMenuTree : loads the menutree from file to menuList array
*******************************************************************/
int loadMenuTree (FILE *fp, char *head)
{
char buf[256];
int i, j, start = 0, count = 0;
fseek(fp, 0, 0);
while (fgets(buf, 255, fp) != NULL)
    {
    j = strlen(buf);
    if (buf[j - 1] == '\n')
        buf[j - 1] = 0;
    if (start)
        {
        if (strlen(buf) == 0)
            break;
        menuList = (int **)realloc(menuList, (count + 1) * sizeof(int *));
```

```
                menuList[count] = (int *)malloc(3 * sizeof(int));
                sscanf(buf, "%d,%d,%d\n", &menuList[count][0],
&menuList[count][1],&menuList[count][2]);
                count++;
                }
        else
                if (!strcmp(head, buf))
                start = 1;
        }
return count ;
}


readini(char * filenm)
{
        FILE * fp;
        char buf[80], key[80], value[80], comment[80];
        int cnt;
        if ((fp=fopen(filenm,"r"))==NULL)
        {
                perror(filenm);
                exit(1);
        }
        while (fgets(buf,79,fp)!=NULL)
        {
        sscanf(buf,"%s %s %s",key,value, comment);
        if (!strcmp(key, "sdoc"))
                        sdoc=strdup(value);
        if (!strcmp(key, "fdoc"))
                        fdoc=strdup(value);
        if (!strcmp(key, "xdoc"))
                        xdoc=strdup(value);
        if (!strcmp(key, "cfg"))
                        cfg=strdup(value);
        if (!strcmp(key, "lcfg"))
                        lcfg=strdup(value);
        if (!strcmp(key, "minprompt"))
                        minPromptCount=atoi(value);
        if (!strcmp(key, "timeout"))
                        timeout=atoi(value);
        }
}
```

interactive.c: This program contains funtions related to user interaction

```
/*********************************************************************
Interactive : function to accept a sentence from the user and then
          generate the response.
---------------------------------------------------------------------
thesaurusFlag = is 1 if there is atleast 1 thesaurus/learned word in query
updateFlag = is set to 1 if the program needs to learn (i.e. main menu was
      selected during the prompt navigation)
interPrompts = Intersection of prompts
unionPrompts = Union of prompts
interUnionPrompts = Intersection of Union
numInter  = number of prompts in InterPrompts
numInterUnion = num of prompts in Intersection of Union
numUnion = num of prompts in Union
numUnknown = num of unknown words
*********************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include "globalvar.h"
#include "arraylib.h"
#include "forms.h"

#define max(a,b) (a > b)? a: b
#define min(a,b) (a < b)? a: b
#define swap(a,b) (a ^= b, b ^= a, a ^= b)

extern int numScore, **scoring;
int updateFlag = 0, learnFlag, numQueryList = 0;
FILE *lf, *pf;
char ** uWList=NULL, *queryTerms[50];
int uWNum;
extern int minPromptCount, timeout;
char query[256], **queryList = NULL;
char *affrmWords[] = { "yes", "right", "correct"};
char *negWords[] = { "no", "neither"};
extern char * fdoc;
int otheFlag = 0;
int unknownWords[20], numQuery = 0, numUnknown;
char **uWords;  // Added this array to facilitate learning wven if lateral shift
int numUW;  // Added this to facilitate learning wven if lateral shift
void sayOther();
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.