```
proj2.c//////////////////////////////////////

/*
Author:         Shane Stacy
Description:    Contains main() and other introduced functions.
                This program uses two singly linked lists to sort identifiers from a C source file.
                The first list will contain unsorted identifiers.
                The second one will contain the sorted identifiers, with the highest occurring
                being towards the front of the list.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "list.h"

static FILE *input;
static FILE *output;
static LINK unsortedList;
static LINK sortedList;

void initializeFiles(char input1[], char output1[]);
void closeFiles();

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

       //  int argc is an int
       //  argv[0] is the program
       //  argv[1] is the first parameter

    unsortedList = makeEmpty();
    sortedList = makeEmpty();

    char *theWord;
    char *otherWord;
    char unfilteredString[100];
    int z = 0;
    int d = 0;
    int status = 0;

    printf("Opening file streams...\n");
    initializeFiles(argv[1], argv[2]);

    while (fgets(unfilteredString, 100, input) != NULL) {

        //  filter the comments
        while (unfilteredString[z] != '\0') {
```

```
        if (unfilteredString[z] == '/') {
            if (unfilteredString[z + 1] == '\0') {
                break;
            }
            else if (unfilteredString[z + 1] == '/') {
                status = 1;
            }
            else if (unfilteredString[z + 1] == '*') {
                status = 2;
                fgets(unfilteredString, 100, input);
            }
        }

        if (unfilteredString[z] == '\"') {
            d = z;
            while (unfilteredString[d] != '\"') {
                unfilteredString[d] = '\0';
                d++;
            }
            d = 0;
        }

        if (status == 1) {
            d = z;
            while (unfilteredString[d] != '\0') {
                unfilteredString[d] = '\0';
                d++;
            }
            d = 0;
            status = 0;
            break;
        }
        else if (status == 2) {
            while (!strstr(unfilteredString, "*/")) {
                fgets(unfilteredString, 100, input);
                if (unfilteredString == NULL)
                    return 1;
            }
        }
        status = 0;
        z++;
    }
    z = 0;

    theWord = strtok(unfilteredString, "\"\'\?\%&><)(;=/\\\"\n\t\v\r:-#}{!][*,.+ ");  //get until terminating character

    printf("Got the word!  Inserting into unsorted list...\n");
    while (theWord != NULL) {  //  while the next word isn't null
```

```
            for (z = 0; theWord[z]; z++)  //  convert the word to lowercase
                theWord[z] = tolower(theWord[z]);
            printf("Converted to lowercase!\n");

            if (isdigit(theWord[0])) {  //  if the first character of theWord is a number, move on to next
word;
                printf("First character was a digit.  Word skipped...\n");
                theWord = strtok(NULL, "\%&><)(;=//\\\"\n\t\v\r:-#}{!][*,.+ ");
                continue;
            }
            else {
                insertFirst(theWord, 1, unsortedList);
                printf("%s inserted!\n", theWord);
                theWord = strtok(NULL, "\%&><)(;=//\\\"\n\t\v\r:-#}{!][*,.+ ");  //  get the next word
                printf("Got the next word! %s\n", theWord);
            }
        }
    }

    //  the unsorted list is now full of identifiers that need to be sorted
    printf("Starting to sort the unsorted list into the sorted list!\n");
    sortTheList(unsortedList, sortedList);  //  sort the unsorted list into a sorted list
    printf("\n\nPrinting the sorted list!\n");
    showList(sortedList, output);  //  show it and print it to a file

    printf("Closing file streams!\n");
    closeFiles();  //  close file streams
    printf("All done!\n");
    return 0;   //  return success
}



//  defines the file streams
void initializeFiles(char input1[], char output1[]) {

       input = fopen(input1, "r");  //  open the input file
       output = fopen(output1, "w");  //  open the output file
}

//  closes file streams
void closeFiles() {

       fclose(input);
       fclose(output);
}
```

```
///////////////////////////////////////////////////////////
list.h
/*
Author:         Shane Stacy
Description:    Contains type definitions and function prototypes for a self-organizing list.
                Only the interfaces for proj2.c are exposed here.
*/

typedef struct Node* LINK;

LINK makeEmpty();
int isEmpty(LINK head);
int insertFirst(char d[], int num, LINK head);
int sortTheList(LINK unsorted, LINK sorted);
void clear(LINK head);
void showList(LINK head, FILE *output);


/////////////////////////////
list.c
/*
Author:         Shane Stacy
Description:    Contains function definitions for a self-organizing list.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//  The first node is a header node that points (next) to the first ELEMENT node.
//  There's a few different ways to remove elements:
//  Doubley Linked Lists (->prev and ->next pointers)
//  Look-Ahead (->next->next)
//  Two Pointers (save into another pointer to be used later)

typedef struct Node* LINK;

int sizeOfList(LINK head);
int searchForWord(LINK head, char word[]);
int searchForCount(LINK head, int num);
int insertFirst(char d[], int num, LINK head);
int deleteFirst(LINK head);
int insertAtIndex(char d[], int num, int elem, LINK head);
int insertLast(char d[], int elem, LINK head);

struct Node {
        char elemString[25];
    int elemInt;
```

```
        LINK next;
        //LINK prev;  doubley linked list
};

//  make an empty list
LINK makeEmpty() {

        LINK head = malloc(sizeof(struct Node));
        head->next = NULL;
        return head;
}

//  is the list empty?
int isEmpty(LINK head) {

        if (head->next == NULL) {
                return 1;
        }
        return 0;
}

//  HELPER:  returns the current size of the list
int sizeOfList(LINK head) {

    LINK curr = head->next;
    int v = 0;

    while (curr != NULL) {
        v++;
        curr = curr->next;
    }
    return v;
}

//  search for the word and return the index
int searchForWord(LINK head, char word[]) {

    if (isEmpty(head)) {
           return 0;
        }

    LINK curr = head->next;
    int i = 1;

    while(curr != NULL) {

        if (strcmp(word, curr->elemString) == 0) {
            return i;
        }
```

```
           curr = curr->next;
        i++;
       }
    return 0;
}

//  HELPER:  search for the first word with the same count and return the index
int searchForCount(LINK head, int num) {

    if (isEmpty(head)) {
           return 0;
       }

    LINK curr = head->next;
    int i = 1;

    while (curr != NULL) {

        if (curr->elemInt > num) {
            curr = curr->next;
        }
        else if (curr->elemInt < num) {
            return i - 1;
        }
        else {
            return i;
        }
        i++;
    }

    return -1;
}

//  HELPER:  insert a word at the front of the list
int insertFirst(char d[], int num, LINK head) {

       LINK ins = (LINK)malloc(sizeof(struct Node));
    LINK temp;

    strcpy(ins->elemString, d);
    ins->elemInt = num;
       temp = head->next;
       head->next = ins;
    ins->next = temp;
       return 1;
}
 //  HELPER:  delete the first element
int deleteFirst(LINK head) {
    LINK temp;

    if (isEmpty(head)) {
       return 0;
    }

    temp = head->next;
    head->next = temp->next;
    free(temp);
    return 1;
}

//  HELPER:  insert a word at the index
int insertAtIndex(char d[], int num, int elem, LINK head) {

  if (isEmpty(head)) {  //  if the list is empty, insert at the front
       insertFirst(d, elem, head);
     return 1;
    }

  if (num == 0) {  //  if the index is 0, insert at the front
       insertFirst(d, elem, head);
     return 1;
    }

  if (sizeOfList(head) + 1 == num) {  //  if the index is 1 beyond the scope, insert at the end
     insertLast(d, elem, head);
     return 1;
  }

  if (sizeOfList(head) < num) {  //  if the index is still beyond the scope, return failure
     return 0;
  }

  int v = sizeOfList(head);
  int i = 1;
  LINK temp = head->next;
  LINK temp2;
  LINK ins = (LINK)malloc(sizeof(struct Node));

  while (i < num) {
     temp = temp->next;
     i++;
  }

  temp2 = temp->next;

  temp->next = ins;
  ins->next = temp2;
```

```
    ins->elemInt = elem;
    strcpy(ins->elemString, d);
    return 1;
}

//  insert a word at the end of the list
int insertLast(char d[], int elem, LINK head) {

    LINK curr = head;

       while(curr->next != NULL) {
           curr = curr->next;
       }

    LINK temp = (LINK)malloc(sizeof(struct Node));
    strcpy(temp->elemString, d);
    temp->elemInt = elem;
    curr->next = temp;
    return 1;
}

//  sort the list
int sortTheList(LINK unsorted, LINK sorted) {

    int i = 0;
    LINK temp;
    LINK temp2;

    temp = unsorted->next;
    temp2 = unsorted->next;
    while (temp != NULL) {
        temp2 = temp;
        if (searchForWord(sorted, temp->elemString) == 0) {
            while (temp2 != NULL) {
                if (strcmp(temp2->elemString, temp->elemString) == 0) {
                    i++;
                    printf("Found %s %d times.\n", temp->elemString, i);
                }
                temp2 = temp2->next;
            }
            int newIndex = searchForCount(sorted, i);
            printf("New index returned %d\n", newIndex);
            if (newIndex == -1) {
                insertLast(temp->elemString, i, sorted);
                printf("Inserting %s at the end.\n", temp->elemString);
            }
            else {
                insertAtIndex(temp->elemString, newIndex, i, sorted);
                printf("Inserting %s at the index %d.\n", temp->elemString, newIndex);
```

```c
            }
            i = 0;
        }
        else {
            printf("%s was already found in the sorted list.  Skipping...\n", temp->elemString);
        }
        printf("Moving on to next word.\n");
        temp = temp->next;
    }
    return 1;
}

//  clear the list
void clear(LINK head) {

        while (!isEmpty(head))
                deleteFirst(head);
}

//  print the list
void showList (LINK head, FILE *output) {

        LINK curr = head->next;
        while (curr != NULL) {
                printf("%s occurred %d times.\n", curr->elemString, curr->elemInt);
         fprintf(output, "%s occurred %d times.\n", curr->elemString, curr->elemInt);
                curr = curr->next;
        }

}
```

//

makefile:

```
test: test1 test2 test3

test1:
        gcc proj2.c list.c

test2:
        ./a.out sample.c output.dat

test3:
```

///

test result:

```
n occurred 20 times.
list occurred 20 times.
printf occurred 17 times.
book occurred 16 times.
numofbooks occurred 14 times.
```

```
int occurred 13 times.
price occurred 12 times.
struct occurred 10 times.
void occurred 8 times.
targetcode occurred 7 times.
code occurred 7 times.
d occurred 6 times.
numofbooksptr occurred 6 times.
index occurred 6 times.
targetprice occurred 6 times.
scanf occurred 5 times.
aprice occurred 5 times.
acode occurred 5 times.
while occurred 4 times.
of occurred 4 times.
float occurred 4 times.
to occurred 4 times.
end occurred 4 times.
if occurred 4 times.
a occurred 3 times.
loadarray occurred 3 times.
? occurred 3 times.
printarray occurred 3 times.
pricesearch occurred 3 times.
codesearch occurred 3 times.
do occurred 3 times.
$ occurred 2 times.
max occurred 2 times.
f occurred 2 times.
enter occurred 2 times.
target occurred 2 times.
not occurred 2 times.
for occurred 2 times.
else occurred 1 times.
include occurred 1 times.
stdio occurred 1 times.
h occurred 1 times.
define occurred 1 times.
main occurred 1 times.
return occurred 1 times.
$0 occurred 1 times.
is occurred 1 times.
allowed occurred 1 times.
books occurred 1 times.
under occurred 1 times.
count occurred 1 times.
findprice occurred 1 times.
found occurred 1 times.
```