**List.c**

#include "list.h"

LINK makeEmpty()

/\* creates an empty list \*/

{

    LINK head = (LINK)malloc(sizeof(struct Node));

    head->next = NULL;

    head->word = '\0';

    return head;

}

int isEmpty(LINK head)

/\* returns true if the list is empty \*/

{

    return (head->next == NULL);

}

int isFull(LINK head)

/\* returns true if the list is full \*/

{

    return FALSE;

}

int deleteFirst(LINK head)

/\* removes the first element from the list and

returns true if the operation is done successfully \*/

{

    LINK temp;

    if (isEmpty(head))

        return FALSE;

    temp = head->next;

    head->next = temp->next;

    free(temp);

    return TRUE;

}

int removeValue(char\* value, LINK head)

/\* removes the first found from the list based off of the search and

returns the value if the operation is done successfully \*/

{

    while (!isEmpty(head)) {

        if (strcmp(head->next->word, value) == 0) {

            int tempValue = head->next->count;

            free(head->next);

            head->next = head->next->next;

            return tempValue;

        }

        else {

            head = head->next;

        }

    }

    return FALSE;

}

int search(char\* value, LINK head)

/\* returns the value of an item with the given search. Then moves it to the front. \*/

{

    int tempValue = removeValue(value, head);

    if (tempValue != FALSE) {

        insertAsFirstWithValue(value, tempValue, head);

        return tempValue;

    }

    else {

        return FALSE;

    }

}

int addValue(char\* value, int add, LINK head) {

    if (search(value, head) != FALSE) {

        head->next->count+=1;

        return TRUE;

    }

    else {

        insertAsFirst(value, head);

        return TRUE;

    }

    return FALSE;

}

int insertAsFirst(char\* value, LINK head)

/\* inserts an element as the first of the list and

returns true if the operation is done successfully \*/

{

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

    temp->word = malloc(strlen(value));

    strcpy(temp->word, value);

    temp->count = 1;

    temp->next = head->next;

    head->next = temp;

    return TRUE;

}

int insertAsFirstWithValue(char\* value, int count, LINK head)

/\* inserts an element as the first of the list with a specific value and

returns true if the operation is done successfully \*/

{

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

    temp->word = malloc(strlen(value));

    strcpy(temp->word, value);

    temp->count = count;

    temp->next = head->next;

    head->next = temp;

    return TRUE;

}

void clear(LINK head)

/\* makes the list empty \*/

{

    int temp;

    while (!isEmpty(head))

        deleteFirst(head);

}

void showList(LINK head)

/\* displays all elements in the list \*/

{

    LINK curr = head->next;

    while (curr != NULL) {

        printf("%s : %d \n", curr->word, curr->count);

        curr = curr->next;

    }

    printf("\n");

}

**List.h**

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE -1

typedef struct Node\* LINK;

struct Node {

    char\* word;

    int count;

    LINK next;

};

LINK makeEmpty();

int isEmpty(LINK head);

int isFull(LINK head);

int deleteFirst(LINK head);

int insertAsFirst(char\* d, LINK head);

int insertAsFirstWithValue(char\* d, int count, LINK head);

int removeValue(char\* value, LINK head);

int search(char\* value, LINK head);

int addValue(char\* value, int add, LINK head);

void clear(LINK head);

void showList(LINK head);

**proj2.c**

#include "list.h"

#include <stdio.h>

void moveFinder(char\* finder, char next);

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

    FILE \*input;

    LINK list = makeEmpty();

    if (argc != 2) {

        printf("This application takes 1 command line argument");

        exit(0);

    }

    input = fopen(argv[1], "r");

    char \*c;

    c = malloc(1);

    char storage[1000] = "";

    char finder[3] = "";

    int commentedMultiline = FALSE;

    int commentedSingleLine = FALSE;

    int inQuotes = FALSE;

    int importName = FALSE;

    while (fgets(c, 2, input) != NULL) {

        moveFinder(&finder, c[0]);

        if (strcmp(finder, "/\*") == 0) {

            commentedMultiline = TRUE;

        }

        if (strcmp(finder, "\*/") == 0) {

            commentedMultiline = FALSE;

        }

        if (strcmp(finder, "//") == 0) {

            commentedSingleLine = TRUE;

        }

        if (finder[1] == '\n') {

            commentedSingleLine = FALSE;

        }

        if (finder[1] == '"') {

            if (inQuotes == FALSE) {

                inQuotes = TRUE;

            }

            else {

                inQuotes = FALSE;

            }

        }

        if (commentedSingleLine == FALSE && commentedMultiline == FALSE && inQuotes == FALSE && importName == FALSE) {

            //If the character is not a letter, underscore, or number

            if (isalpha(c[0]) == 0 && isdigit(c[0]) == 0 && c[0] != '\_') {

                if (strlen(storage) > 0) {

                    addValue(storage, 1, list);

                    storage[0] = '\0';

                }

            }

            else {

                if (c[0] != '\t') {

                    if (strlen(storage) == 0) {

                        if (isalpha(c[0]) != 0) {

                            strcat(storage, c);

                        }

                    }

                    else {

                        strcat(storage, c);

                    }

                }

            }

        }

    }

    showList(list);

}

void moveFinder(char\* finder, char next) {

    if (strlen(finder) == 2) {

        finder[0] = finder[1];

        finder[1] = next;

    }

    else if (strlen(finder) == 1){

        finder[1] = next;

    }

    else if (strlen(finder) == 0) {

        finder[0] = next;

    }

}

**Makefile**

main: proj2.o list.o

    gcc proj2.o list.o -o main

list.o: list.c list.h

proj2.o: proj2.c list.h

**Program Output**

printf : 17

targetCode : 7

while : 4

price : 7

else : 1

if : 4

numOfBooks : 14

list : 19

FindPrice : 1

scanf : 5

do : 3

float : 4

int : 13

Book : 10

struct : 10

CodeSearch : 3

void : 8

targetPrice : 6

Count : 1

PriceSearch : 3

index : 6

code : 3

for : 1

printArray : 3

aCode : 5

numOfBooksPtr : 6

aPrice : 5

loadArray : 3

return : 1

MAX : 2

main : 1

define : 1

h : 1

stdio : 1

include : 1