מטלה מספר 10 – רשימה מקושרת דו כיוונית

## קוד שאלה 1:

#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INCLUDES & DEFINES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

#define WORD\_LEN 30

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* STRUCT , UNION & ENUM DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//doubly linked list NODE.

typedef struct w\_list {

struct w\_list \*next;

struct w\_list \*prev;

char word[WORD\_LEN];

}W\_LIST;

//envelope struct for doubly linked list.

typedef struct DLL {

W\_LIST\* head;

W\_LIST\* tail;

int len;

int numOfInserts;

int numOfRemoves;

}DLL;

//used for fault message prints and fault exit routine.

typedef enum ERROR\_MSG{

MEMORY,

NULL\_PTR,

NOT\_NULL\_PTR,

MSG\_LEN,

PTR\_NOT\_EQL

}ERROR\_MSG;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN FUNCTION DECLARATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* allocates memory for a new DLL Envelope and NULLS all pointers.

\*

\* @param - dList = NULL pointer to DLL.

\* @asserts - dList is NOT NULL.

\* @return - allocated pointer to DLL if successful.

,exits otherwise.

\*/

DLL\* dllInit(DLL\* dList);

DLL\* dllFree(DLL\* dList);

DLL\* add2head(DLL\* dList,char\* word);

DLL\* add2tail(DLL\* dList,char\* word);

DLL\* deleteCurr(DLL\* dlist, W\_LIST\*\* curr);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AUX FUNCTION DECLARATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* allocates memory for a new dll NODE and populates it.

\*

\* @param - string data for populating node of max length 29.

\* @asserts - word is not void

\* @return - a newly allocated dll node if successful.

\* ,exits otherwise.

\*/

W\_LIST\* createNode(char\* word);

/\*\*

\* - hardFault reports errors to error log and exits.

\*

\* @param - error message type.

\* @return - prints to errorLog.txt. all faults exit with value 1.

\*/

void hardFault(ERROR\_MSG err, char\* fCall);

void menu(DLL\* list);

void nextWord(DLL\* list, W\_LIST\*\* curr);

void prevWord(DLL\* list, W\_LIST\*\* curr);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void) {

DLL\* newList = NULL;

newList = dllInit(newList);

newList = add2tail(newList,"this");

newList = add2tail(newList,"is");

newList = add2tail(newList,"very");

newList = add2tail(newList,"boring");

menu(newList);

return EXIT\_SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN FUNCTION IMPLEMENTATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DLL\* dllInit(DLL\* dList) {

if (dList) hardFault(NOT\_NULL\_PTR,"dllInit");

dList = (DLL\*)calloc(1, sizeof(DLL));

if (!dList) hardFault(MEMORY,"dllInit");

return dList;

}

DLL\* dllFree(DLL\* dList) {

if (!dList) return NULL;

W\_LIST\* temp = dList->head;

W\_LIST\* forDestruct = NULL;

while (temp) {

forDestruct = temp;

temp = temp->next;

free(forDestruct);

}

free(dList);

return NULL;

}

DLL\* add2head(DLL\* dList,char\* word) {

if (!dList) hardFault(NULL\_PTR,"add2head");

W\_LIST\* newNode = createNode(word);

if (dList->len == 0) {

if (dList->head != NULL || dList->tail != NULL) hardFault(NOT\_NULL\_PTR,"add2head");

dList->head = newNode;

dList->tail = newNode;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

if (dList->len == 1) {

if (dList->head != dList->tail) hardFault(PTR\_NOT\_EQL,"add2head");

dList->head = newNode;

dList->head->next = dList->tail;

dList->tail->prev = dList->head;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

newNode->next = dList->head;

dList->head->prev = newNode;

dList->head = newNode;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

DLL\* add2tail(DLL\* dList,char\* word) {

if (!dList) hardFault(NULL\_PTR,"add2tail");

W\_LIST\* newNode = createNode(word);

if (dList->len == 0) {

if (dList->head != NULL || dList->tail != NULL) hardFault(NOT\_NULL\_PTR,"add2tail");

dList->head = newNode;

dList->tail = newNode;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

if (dList->len == 1) {

if (dList->head != dList->tail) hardFault(PTR\_NOT\_EQL,"add2tail");

dList->tail = newNode;

dList->head->next = dList->tail;

dList->tail->prev = dList->head;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

newNode->prev = dList->tail;

dList->tail->next = newNode;

dList->tail = newNode;

++(dList->len);

++(dList->numOfInserts);

return dList;

}

DLL\* deleteCurr(DLL\* dlist, W\_LIST\*\* curr) {

if (!dlist) hardFault(NULL\_PTR,"deleteCurr");

if (!(\*curr)) { printf("cannot delete from an empty list.\n"); return dlist; }

if ((\*curr) == dlist->head) {

if ((\*curr)->next == dlist->tail) {

free((\*curr));

--(dlist->len);

++(dlist->numOfRemoves);

dlist->head = dlist->tail;

dlist->head->prev = NULL;

(\*curr) = dlist->head;

return dlist;

}

dlist->head = (\*curr)->next;

dlist->head->prev = NULL;

free((\*curr));

--(dlist->len);

++(dlist->numOfRemoves);

(\*curr) = dlist->head;

return dlist;

}

if ((\*curr) == dlist->tail) {

if ((\*curr)->prev == dlist->head) {

free((\*curr));

--(dlist->len);

++(dlist->numOfRemoves);

dlist->tail = dlist->head;

dlist->head->next = NULL;

(\*curr) = dlist->tail;

return dlist;

}

dlist->tail = (\*curr)->prev;

dlist->tail->next = NULL;

free((\*curr));

--(dlist->len);

++(dlist->numOfRemoves);

(\*curr) = dlist->tail;

return dlist;

}

(\*curr)->prev->next = (\*curr)->next;

(\*curr)->next->prev = (\*curr)->prev;

W\_LIST\* temp = (\*curr)->prev;

free(\*curr);

(\*curr) = temp;

--(dlist->len);

++(dlist->numOfRemoves);

return dlist;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AUX FUNCTION IMPLEMENTATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

W\_LIST\* createNode(char\* word) {

if (!word) hardFault(NULL\_PTR,"createNode");

if (strlen(word) >= WORD\_LEN) hardFault(MSG\_LEN,"createNode");

W\_LIST\* newNode = NULL;

newNode = (W\_LIST\*)calloc(1,sizeof(W\_LIST));

if (!newNode) hardFault(MEMORY,"createNode");

strcpy(newNode->word,word);

return newNode;

}

void hardFault(ERROR\_MSG err, char\* fCall) {

printf("critical error. program will terminate.\n\

check Log file \"errorLog.txt\" for details.\n");

FILE\* errLog = NULL;

errLog = fopen("errorLog.txt", "at");

if (!errLog) {

printf("\ncannot create error log file.\n");

exit(EXIT\_FAILURE);

}

time\_t currTime;

currTime = time(NULL);

fputs(ctime(&currTime),errLog);

switch (err)

{

case MEMORY:

fprintf(errLog,"critical memory allocation error - %s.\n\n",fCall);

exit(EXIT\_FAILURE);

case NULL\_PTR:

fprintf(errLog,"NULL pointer access error - %s.\n\n",fCall);

exit(EXIT\_FAILURE);

case NOT\_NULL\_PTR:

fprintf(errLog,"pointer override with NULL error - %s.\n\n",fCall);

exit(EXIT\_FAILURE);

case MSG\_LEN:

fprintf(errLog,"word length exceeds asserted max - %s.\n\n",fCall);

exit(EXIT\_FAILURE);

case PTR\_NOT\_EQL:

fprintf(errLog,"pointers not equal error - %s.\n\n",fCall);

exit(EXIT\_FAILURE);

default:

fprintf(errLog,"unknown error.\n\n");

exit(EXIT\_FAILURE);

}

fclose(errLog);

}

void menu(DLL\* list) {

char choice = 'N', word[WORD\_LEN];

W\_LIST\* curr = list->head;

printf("Menu:\n\

\*\*\*\*\*\*\n\

N - next word in list.\n\

P - previous word in list.\n\

D - delete current word.\n\

E - delete word at position X (input) in list.\n\

A - add word to list after current word position.\n\

B - add word to list head..\n\

C - add word to list tail.\n\

F - add word at position X (input) in list\n\

G - print word list.\n\

S - dump word list to text log.\n\

Q - exit.\n\n");

printf("current word is \"%s\".\n\n",curr->word);

while (choice != 'Q')

{

fflush(stdin);

printf("Enter next menu choice: ");

scanf("%s", &choice);

switch (choice)

{

case 'N':

nextWord(list,&curr);

break;

case 'P':

prevWord(list,&curr);

break;

case 'D':

list = deleteCurr(list,&curr);

break;

case 'E':

break;

case 'A':

break;

case 'B':

printf("enter word of length < %d to insert: ",WORD\_LEN);

scanf("%s",word);//does not check overflow!!!

list = add2head(list,word);

break;

case 'C':

printf("enter word of length < %d to insert: ",WORD\_LEN);

scanf("%s",word);//does not check overflow!!!

list = add2tail(list,word);

break;

case 'F':

break;

case 'G':

break;

case 'S':

break;

case ('Q'):

list = dllFree(list);

exit(EXIT\_SUCCESS);

break;

default: break;

}

}

return;

}

void nextWord(DLL\* list, W\_LIST\*\* curr) {

if (!(\*curr)->next) { printf("list tail reached...\n"); return; }

\*curr = (\*curr)->next;

printf("current word is \"%s\"\n",(\*curr)->word);

return;

}

void prevWord(DLL\* list, W\_LIST\*\* curr) {

if (!(\*curr)->prev) { printf("list head reached...\n"); return; }

\*curr = (\*curr)->prev;

printf("current word is \"%s\"\n",(\*curr)->word);

return;

}

### פלט שאלה 1:

לא הספקתי לכתוב מספיק מהקוד כדי שיהיה יותר מדי מה להראות. מחכה לאישור להארכת הגשה מאביב כהנא המתרגל שאחראי עלי.