Pair Programming 5 Turn In

Name: \_\_\_\_\_Braden Bell\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Username: \_\_\_c1020a01\_\_\_\_\_\_\_\_

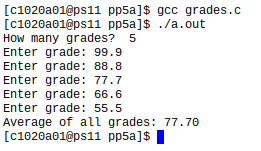
Partner name: \_\_\_\_\_Thomas Snyder\_\_\_\_\_\_\_\_\_\_ Partner username: \_\_\_\_\_c1020a10\_\_\_\_\_\_\_

\_X\_\_ I certify that my partner worked with me on this assignment.

SCORE: \_\_\_\_\_\_\_\_\_\_\_\_ (to be filled in by instructor)

5a (5 points)

**[PASTE THE SCREEN SHOT OF THE TESTS WITH A WHITE BACKGROUND FOR PAIR PROGRAMMIG 5A HERE]**



**[PASTE CODE FOR PAIR PROGRAMMING 5A HERE – NOT A SCREEN SHOT. DOWNLOAD THE UPDATED gradesStart.c. OPEN IT WITH A TEXT EDITOR LIKE NOTEPAD++ OR WORDPAD, COPY THE CODE, PASTE IT HERE. MAKE SURE IT IS SINGLE SPACED AND USE A COURIER NEW FONT]**

/\* File: grades.c

\* Author: Cindy Arnold

\* Description: demonstrates dynamically allocates array

\*/

#include <stdio.h>

#include <stdlib.h>

int main() {

int count, i;

// ADD CODE HERE TO declare pointer to grades (doubles)

double \*grades;

double sum = 0, average = 0;

// as user how many grades

printf( "How many grades? " );

scanf( "%d", &count );

// ADD CODE HEE TO dynamically allocate count grades

grades = (double\*) malloc(sizeof(double)\*count);

// ADD CODE HERE TO get the grades

i = 0;

while (i < count) { //User populates grades array until it is full

printf("Enter grade: ");

scanf( "%lf", &grades[i]);

i++;

}

// ADD CODE HERE TO compute and print the sum and average

for (i = 0; i < count; i++) { //Calculate sum

sum += grades[i];

}

average = (sum /(double)count); //Calculate average

//printf("Sum of all grades: %0.2lf\n", sum);

printf("Average of all grades: %0.2lf\n", average); //Print average

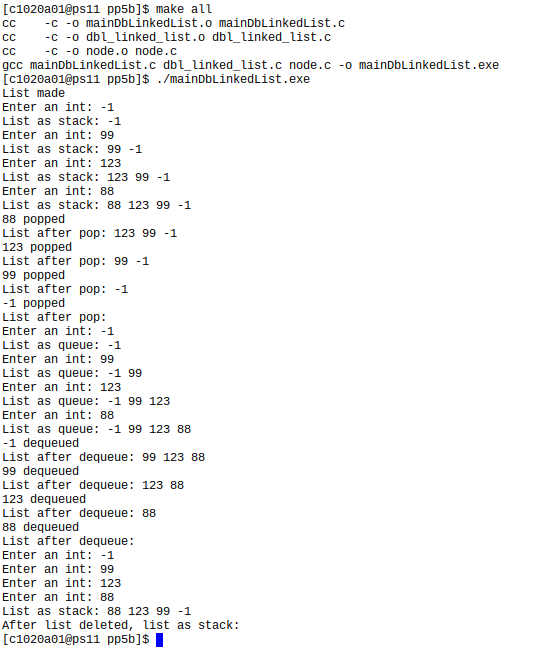
free(grades); //Deallocate dynamic memory to prevent memory leaks

return 0;

}

5b (5 points)

**[PASTE THE SCREEN SHOT OF THE TESTS WITH A WHITE BACKGROUND FOR PAIR PROGRAMMIG 5B HERE]**



**[PASTE CODE FOR PAIR PROGRAMMING 5B HERE – NOT A SCREEN SHOT. DOWNLOAD node.h, node.c, dbl\_linked\_list.h, AND dbl\_linked\_list.c. OPEN THEM WITH A TEXT EDITOR LIKE NOTEPAD++ OR WORDPAD, COPY THE CODE, PASTE IT HERE. MAKE SURE IT IS SINGLE SPACED AND USE A COURIER NEW FONT]**

**node.h:**

#ifndef NODE\_H

#define NODE\_H

typedef struct node\_t {

int i;

struct node\_t\* nextPtr;

struct node\_t\* prevPtr;

} node\_t;

node\_t\* initNode(int);

#endif

**node.c:**

#include <stdio.h>

#include <stdlib.h>

#include "node.h"

/\* initNode: initializes a node set up for a doubly linked list

\* Parameters:

\* i = the data to be stored in the node

\* Returns: a pointer to the node

\*/

node\_t\* initNode(int data) {

node\_t\* nodePtr;

nodePtr = malloc(sizeof(node\_t));

nodePtr -> i = data;

nodePtr -> nextPtr = NULL;

nodePtr -> prevPtr = NULL;

return nodePtr;

}

**dbl\_linked\_list.h:**

#ifndef DBL\_LINKED\_LIST\_H

#define DBL\_LINKED\_LIST\_H

#include "node.h"

typedef struct {

node\_t\* headPtr;

node\_t\* tailPtr;

int count;

} dbl\_linked\_list\_t;

void createList(dbl\_linked\_list\_t\*);

void insertNode(dbl\_linked\_list\_t\*, node\_t\*);

node\_t\* popNode(dbl\_linked\_list\_t\*);

node\_t\* dequeueNode(dbl\_linked\_list\_t\*);

void deleteList(dbl\_linked\_list\_t\*);

void traverseStack(const dbl\_linked\_list\_t\*);

void traverseQueue(const dbl\_linked\_list\_t\*);

#endif

**dbl\_linked\_list.c:**

#include <stdio.h>

#include <stdlib.h>

#include "dbl\_linked\_list.h"

/\* createList: initializes the doubly linked list

\* Parameters:

\* listPtr - pointer to the list

\* Returns: nothing

\*/

void createList(dbl\_linked\_list\_t\* listPtr) {

if (listPtr != NULL) {

listPtr -> count = 0;

listPtr -> headPtr = NULL;

listPtr -> tailPtr = NULL;

}

}

/\* insertNode: Inserts a new node in the first slot of the doubly linked list

\* Parameters:

\* listPtr - pointer to the list

\* nPtr - points to the node

\* Returns: nothing

\*/

void insertNode(dbl\_linked\_list\_t\* listPtr, node\_t\* nPtr) {

if (listPtr != NULL) {//Make sure the list has been created by createList

listPtr -> count++;

if(listPtr -> headPtr == NULL ) { //If listPtr's list is empty

//printf("headPtr was NULL\n");

listPtr -> headPtr = nPtr;

listPtr -> tailPtr = nPtr;

}

else { //If listPtr's list is not empty

//printf("headPtr was not NULL\n");

nPtr -> nextPtr = listPtr -> headPtr;

listPtr -> headPtr -> prevPtr = nPtr;

listPtr -> headPtr = nPtr;

}

}

}

/\* popNode: pops the first node out of the doubly linked list

\* Parameters:

\* listPtr - pointer to the list

\* Returns: a pointer to the node that was popped

\*/

node\_t\* popNode(dbl\_linked\_list\_t\* listPtr) {

if(listPtr != NULL && listPtr -> headPtr != NULL) { //If the list is not empty and it was initialized

listPtr -> count--; //A. Decriment count

node\_t\* nodePtr = listPtr -> headPtr; //B. nodePtr = the node at the end of the list

listPtr -> headPtr = listPtr -> headPtr -> nextPtr; //C. Set headPtr to point to next in the list

if (listPtr -> headPtr != NULL) { //If the list is still not empty

listPtr -> headPtr -> prevPtr = NULL; //Set headPtr's previous pouinter to NULL

}

else { //If the list is empty:

listPtr -> tailPtr = NULL; //set tailPtr to NULL

}

nodePtr -> prevPtr = NULL; //Set nodeptr's prev and next ptrs to NULL since the node's not in the list anymore

nodePtr -> nextPtr = NULL;

return nodePtr; //Return nodePtr

}

else

return NULL; //Return NULL

}

/\* createList: Prints the data in the doublely linked list as if it were a list

\* Parameters:

\* listPtr - pointer to the list

\* Returns: nothing

\*/

void traverseStack(const dbl\_linked\_list\_t\* listPtr) {

if (listPtr != NULL) { //If list has been initialized

node\_t\* curPtr = listPtr -> headPtr; //curPtr = list's headPtr

while(curPtr != NULL) { //While curPtr isn't NULL

printf("%d ", curPtr -> i); //Print curPtr's data

curPtr = curPtr -> nextPtr; //Update curPtr to point to the next node

}

}

}

/\* dequeueNode: initializes the list

\* Parameters:

\* listPtr - pointer to the list

\* Returns: a pointer to the item dequeued

\*/

node\_t\* dequeueNode(dbl\_linked\_list\_t\* listPtr) {

if(listPtr != NULL && listPtr -> headPtr != NULL) { //If the list is not empty and it was initialized

listPtr -> count--; //A. Decriment count

node\_t\* nodePtr = listPtr -> tailPtr; //B. nodePtr = the node at the end of the list

listPtr -> tailPtr = listPtr -> tailPtr -> prevPtr; //C. Set headPtr to point to next in the list

if (listPtr -> tailPtr != NULL) { //If the list is still not empty

listPtr -> tailPtr -> nextPtr = NULL; //Set headPtr's previous pouinter to NULL

}

else { //If the list is empty:

listPtr -> headPtr = NULL; //set tailPtr to NULL

}

nodePtr -> prevPtr = NULL; //Set nodeptr's prev and next ptrs to NULL since the node's not in the list anymore

nodePtr -> nextPtr = NULL;

return nodePtr; //Return nodePtr

}

else

return NULL; //Return NULL

}

/\* traverseQueue: prints items in the dynamically linked list as if it were a queue

\* Parameters:

\* listPtr - pointer to the list

\* Returns: nothing

\*/

void traverseQueue(const dbl\_linked\_list\_t\* listPtr) {

if (listPtr != NULL) { //If list has been initialized

node\_t\* curPtr = listPtr -> tailPtr; //curPtr = list's tailPtr

while(curPtr != NULL) { //While curPtr isn't NULL

printf("%d ", curPtr -> i); //Print curPtr's data

curPtr = curPtr -> prevPtr; //Update curPtr to point to the next node

}

}

}

/\* deleteList: Deletes the doubly linked list

\* Parameters:

\* listPtr - pointer to the list

\* Returns: nothing

\*/

void deleteList(dbl\_linked\_list\_t\* listPtr) {

if (listPtr != NULL && listPtr -> headPtr != NULL) { //If the list exists and is not empty

node\_t\* curPtr = listPtr -> headPtr; //curPtr = the list's headPtr

while(curPtr != NULL) { //While curPtr isn't null:

listPtr -> headPtr = curPtr -> nextPtr; //headPtr = curPtr's nextPtr

free(curPtr); //Deallocate curPtr's memory

curPtr = listPtr -> headPtr; //Update curPtr

}

listPtr -> tailPtr = NULL; //Set list's tail pointer to NULL

listPtr -> count = 0; //Set list's count to 0

}

}