שם: אביתר קורקוס תז: 207983198  
שם: נגה חסיד תז: 207486598

מטלה 3

קישור ל- GitHub Repository עבור המטלה: <https://github.com/EvyaKor/Adv_C_Exe3>

**1. Stack.c - מחסנית**

\*חתימות הפונקציות נמצאות בקובץ Stack.h בפרויקט

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Stack.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Stack ADT Implementation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initStack(Stack\* s)

{

s->head = NULL;

}

void destroyStack(Stack\* s)

{

charNode\* temp;

while (s->head != NULL)

{

temp = s->head;

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

free(temp);

}

}

void push(Stack\* s, char data)

{

charNode\* newitem = (charNode\*)malloc(sizeof(charNode));

if (!newitem)

{

printf("push: Memory allocation failed\n");

exit(0);

}

newitem->data = data;

s->head = addToHead(s->head, newitem); // Recall private function

}

char pop(Stack\* s)

{

char res;

res = s->head->data;

removeItem(&(s->head)); //pop current head from the stack

return(res);

}

int isEmptyStack(const Stack\* s)

{

if (!s->head)

{

return 1;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions using stacks - Implementation/definition \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void flipBetweenHashes(const char\* sentence)

{

// In case string is empty

if (!sentence)

{

printf("No sentence detected, String must not be empty!\n");

return;

}

// In case string is not empty

Stack Temp; // Create auxiliary stack

initStack(&Temp); // Init stack

int size = strlen(sentence);

char\* resSentence = (char\*)calloc(size, sizeof(char) + 1); // Init string for result

if (!resSentence)

{

printf("Memory allocation failed!\n");

exit(0);

}

char\* ptr2sentence = sentence; // Pointer to read from string

char\* ptr2res = resSentence; // Pointer to write to result

while (\*ptr2sentence != '\0') // Read from string

{

if (\*ptr2sentence == '#') // Reached the char '#'

{

ptr2sentence++;

while (\*ptr2sentence != '#')

{ // Until reaches next '#', push each char to stack

push(&Temp, \*ptr2sentence);

ptr2sentence++;

}

while (!isEmptyStack(&Temp))

{ // Until stack is empty, pop each char to result string

\*ptr2res++ = pop(&Temp);

}

ptr2sentence++;

}

strncpy(ptr2res++, ptr2sentence++, sizeof(char)); // Copy string to result, char by char

}

printf("The new string is:\n%s\n", resSentence);

free(resSentence); // Free result string

destroyStack(&Temp); // Delete auxiliary stack

}

int isPalindrome(Stack\* s)

{

// In case stack does not exist

if (s == NULL)

{

printf("Uninitialized stack!\n");

return 0;

}

// In case stack is empty

if (isEmptyStack(s))

{

return 1;

}

// In case stack is not empty

Stack Temp; // Create auxiliary stack

initStack(&Temp); // Init stack

int midsize = howManyInStack(s) / 2; // Get the size of half the stack

charNode\* ptr1 = s->head; // pointer to read from stack

charNode\* ptr2; // pointer to read from auxiliary stack

while (ptr1 != NULL)

{

push(&Temp, ptr1->data); // Copy stack to auxiliary stack in reverse

ptr1 = ptr1->next;

}

ptr1 = s->head;

ptr2 = Temp.head;

for (int i = 0; i < midsize; i++) // Loop to check palindrom, works for both odd and even size options

{

if (ptr1->data != ptr2->data)

{

return 0;

}

ptr1 = ptr1->next;

ptr2 = ptr2->next;

}

destroyStack(&Temp); // Delete auxiliary stack

return 1;

}

void rotateStack(Stack\* s, int n)

{

// In case stack does not exist

if (s == NULL)

{

printf("Uninitialized stack!\n");

return;

}

// In case stack is empty

if (isEmptyStack(s))

{

return;

}

// In case n is negative

if (n < 0)

{

return;

}

// In case n is bigger than stack size

int size = howManyInStack(s);

if (n > size)

{

return;

}

// In case stack is not empty and n is valid

Stack Temp1; // Create auxiliary stack 1

initStack(&Temp1); // Init stack 1

Stack Temp2; // Create auxiliary stack 2

initStack(&Temp2); // Init stack 2

char tempchar;

for (int i = 0; i < (size - n); i++) // pop size-n items from stack and push to Temp1

{

tempchar = pop(s);

push(&Temp1, tempchar);

}

while (s->head != NULL) // pop the rest of the items from stack and push to Temp2

{

tempchar = pop(s);

push(&Temp2, tempchar);

}

// now stack is empty

while (Temp1.head != NULL) // pop all items from Temp1 and push back to stack

{

tempchar = pop(&Temp1);

push(s, tempchar);

}

while (Temp2.head != NULL) // pop all items from Temp1 and push back to stack

{

tempchar = pop(&Temp2);

push(s, tempchar);

}

destroyStack(&Temp1); // Delete auxiliary stack 1

destroyStack(&Temp2); // Delete auxiliary stack 2

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Private functions for Linked List \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Adds new List Item to head of list

charNode\* addToHead(charNode\* head, charNode\* toAdd)

{

toAdd->next = head;

head = toAdd;

return head;

}

// Deletes the first List Item from list

void removeItem(charNode\*\* head)

{

if (\*head == NULL) return; //if stack empty

charNode\* temp = \*head;

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

free(temp);

}

// Prints Linked List

void printList\_Stack(charNode\* head)

{

charNode\* temp;

temp = head;

while (temp != NULL)

{

printf("%c\n", temp->data);

temp = temp->next;

}

}

// Counts how many in stack

int howManyInStack(Stack\* s)

{

int counter = 0;

charNode\* temp;

temp = s->head;

while (temp != NULL)

{

counter++;

temp = temp->next;

}

return counter;

}

**Main.c - לבדיקת הפונקציות**

#include <stdio.h>

#include "Stack.h"

#include "Queue.h"

void main()

{

printf("\n############################### Testing part 1 - stack ###############################\n");

// Testing stack funtions

printf("\n==============Testing stack functions==============\n");

printf("\n----1st test(initStack and push and printList\_Stack and howManyInStack)----\n");

Stack Test;

initStack(&Test); //test initStack

push(&Test, 'A'); //test push + addToHead

push(&Test, 'B');

push(&Test, 'C');

push(&Test, 'D');

push(&Test, 'C');

push(&Test, 'B');

push(&Test, 'A');

printList\_Stack(Test.head); //test printList\_Stack

printf("There are %d items in stack\n", howManyInStack(&Test)); //test howManyInStack

/\*

printf("\n----next test(pop)----\n");

char tav = pop(&Test); //test pop + removeItem

printList\_Stack(Test.head);

printf("\nThe tav that was poped is: %c\n", tav);

printf("\n----next test(isEmptyStack and destroyStack)----\n");

printf("Before destroy: %d\n", isEmptyStack(&Test)); //test isEmptyStack

destroyStack(&Test); //test destroyStack

printf("After destroy: %d\n", isEmptyStack(&Test)); //test isEmptyStack

\*/

printf("\n===================================================\n");

// Testing flipBetweenHashes

printf("\n==============Testing flipBetweenHashes==============\n");

printf("\n1st test: ");

const char\* sentence1 = "Remem#reb# thi#carp s#tice";

flipBetweenHashes(sentence1);

printf("\nnext test: ");

const char\* sentence2 = "#meR#embe#siht r# practice";

flipBetweenHashes(sentence2);

printf("\nnext test(In case string is empty): ");

flipBetweenHashes(NULL);

printf("\n===================================================\n");

// Testing isPalindrome

printf("\n==============Testing isPalindrome==============\n");

printf("\n----1st test(In case stack does not exist)----\n");

printf("%d\n", isPalindrome(NULL)); //In case stack does not exist

printf("\n----next test(Check if palindrom)----\n");

printf("%d\n", isPalindrome(&Test));

printf("\n===================================================\n");

// Testing rotateStack

printf("\n==============Testing rotateStack==============\n");

printf("\n----1st test(In case stack does not exist)----\n");

rotateStack(NULL, 2); // In case stack does not exist

printList\_Stack(Test.head);

printf("\n----next test(In case n is negative)----\n");

rotateStack(&Test, -3); // n=-3 (for test)

printList\_Stack(Test.head);

printf("\n----next test(In case n is bigger than stack size)----\n");

rotateStack(&Test, 8); // n=8 (for test)

printList\_Stack(Test.head);

printf("\n----next test(Check if rotateStack worked)----\n");

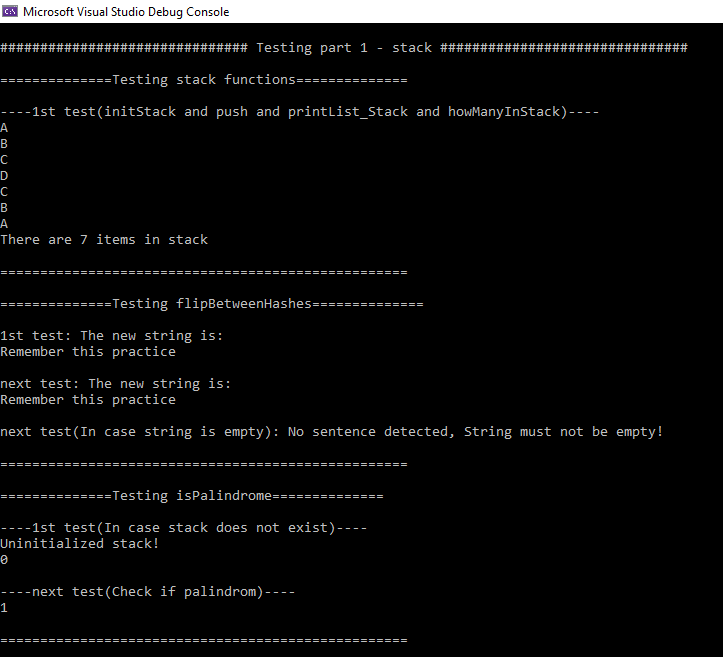
rotateStack(&Test, 2); // n=2 (for test)

printList\_Stack(Test.head);

printf("\n===================================================\n");

}

**פלט:**





**2. Queue.c - תור**

\*חתימות הפונקציות נמצאות בקובץ Queue.h בפרויקט

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Queue.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Queue ADT Implementation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initQueue(Queue\* q)

{

q->head = q->tail = NULL;

}

void destroyQueue(Queue\* q)

{

intNode\* temp;

while (q->head != NULL)

{

temp = q->head;

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

free(temp);

}

q->head = q->tail = NULL;

}

void enqueue(Queue\* q, unsigned int data)

{

intNode\* newitem = (intNode\*)malloc(sizeof(intNode));

if (!newitem)

{

printf("push: Memory allocation failed\n");

exit(0);

}

newitem->data = data;

newitem->next = NULL;

// In case queue is empty

if (isEmptyQueue(q))

{

q->head = q->tail = newitem;

return;

}

// In case queue is not empty

q->tail->next = newitem;

q->tail = newitem;

}

unsigned int dequeue(Queue\* q)

{

unsigned res;

res = q->head->data;

intNode\* temp = q->head;

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

if (q->head == NULL) // if queue gets empty both head and tail NULL

{

q->tail = NULL;

}

free(temp);

return res;

}

int isEmptyQueue(const Queue\* q)

{

if (q->head == NULL && q->tail == NULL)

{

return 1;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions using Queues - Implementation/definition \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void rotateQueue(Queue\* q)

{

// In case queue does not exist

if (q == NULL)

{

printf("Uninitialized queue!\n");

return;

}

// In case queue is empty

if (isEmptyQueue(q))

{

return;

}

// In case queue is not empty

Queue Temp; // Create auxiliary queue

initQueue(&Temp); // Init queue

unsigned int num;

while (q->head != q->tail) //dequeue all items besides the last item(tail) from queue and enqueue to Temp

{

num = dequeue(q);

enqueue(&Temp, num);

}

while (Temp.head != NULL) //dequeue all items from Temp and enqueue back to queue

{

num = dequeue(&Temp);

enqueue(q, num);

}

destroyQueue(&Temp); // Delete auxiliary queue

}

void cutAndReplace(Queue\* q)

{

// In case queue does not exist

if (q == NULL)

{

printf("Uninitialized queue!\n");

return;

}

int size = howManyInQueue(q);

int avarage = 0;

intNode\* ptr = q->head; // pointer to read from queue

if (size % 2 != 0) // in case size is odd

{

while (ptr != NULL)

{

avarage = avarage + ptr->data;

ptr = ptr->next;

}

avarage = avarage / size; // now avarage is the avarage of the items in the queue

enqueue(q, avarage);

}

// now size of queue is even

size = howManyInQueue(q); // update size of queue

Queue Temp; // Create auxiliary queue

initQueue(&Temp); // Init queue

unsigned int num;

for (int i = 0; i < (size / 2); i++) //dequeue half of the queue and enqueue to queue

{

num = dequeue(q);

enqueue(&Temp, num);

}

//reversing the tail half of the queue using 3 pointers

intNode\* temp\_curr = q->head;

intNode\* temp\_next = NULL;

intNode\* temp\_prev = NULL;

while (temp\_curr != NULL)

{

temp\_next = temp\_curr->next;

temp\_curr->next = temp\_prev;

temp\_prev = temp\_curr;

temp\_curr = temp\_next;

}

q->tail = q->head;

q->head = temp\_prev;

while (Temp.head != NULL) //dequeue all items from Temp and enqueue back to queue

{

num = dequeue(&Temp);

enqueue(q, num);

}

destroyQueue(&Temp); // Delete auxiliary queue

}

void sortKidsFirst(Queue\* q)

{

// In case queue does not exist

if (q == NULL)

{

printf("Uninitialized queue!\n");

return NULL;

}

// In case queue is empty

if (isEmptyQueue(q))

{

return NULL;

}

unsigned int num = 0;

Queue Temp; // Create auxiliary queue

initQueue(&Temp); // Init queue

intNode\* temp\_read = q->head; // pointer to read from queue

while (q->head != NULL) //dequeue all itmes from queue and enqueue to Temp

{

num = dequeue(q);

enqueue(&Temp, num);

}

intNode\* index; // pointer to hold minimum value index

while (!isEmptyQueue(&Temp))

{

index = minValueIndex(&Temp); // get minimum value index

moveIndexToHead(&Temp, index); // move it to head of Temp

num = dequeue(&Temp); // dequeue from Temp and enqueue back to q

enqueue(q, num);

}

destroyQueue(&Temp); // Delete auxiliary queue

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Private functions for Linked List \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Prints Linked List

void printList\_Queue(intNode\* head)

{

intNode\* temp;

temp = head;

while (temp != NULL)

{

printf("%d\n", temp->data);

temp = temp->next;

}

}

// Counts how many in queue

int howManyInQueue(Queue\* q)

{

int counter = 0;

intNode\* temp;

temp = q->head;

while (temp != NULL)

{

counter++;

temp = temp->next;

}

return counter;

}

// Finds the index of the min value in queue

intNode\* minValueIndex(Queue\* q)

{

// In case queue does not exist

if (q == NULL)

{

printf("Uninitialized queue!\n");

return NULL;

}

// In case queue is empty

if (isEmptyQueue(q))

{

return NULL;

}

intNode\* ptr\_read = q->head; // pointer to read from queue

intNode\* ptr\_min = q->head; // pointer to hold min value index

unsigned int min = 4294967295; // maximum unsigned integer possible

while (ptr\_read != NULL)

{

if (ptr\_min->data > ptr\_read->data)

{

ptr\_min = ptr\_read;

}

ptr\_read = ptr\_read->next;

}

// now ptr\_min points to the smallest number in the queue

return ptr\_min;

}

// Moves given index to head of queue

void moveIndexToHead(Queue\* q, intNode\* index)

{

// In case queue does not exist

if (q == NULL)

{

printf("Uninitialized queue!\n");

return;

}

// In case queue is empty

if (isEmptyQueue(q))

{

return;

}

intNode\* ptr = q->head; // pointer to read from queue

while ((ptr != NULL) && (ptr != index)) // moves head of queue to tail until reaches index

{

int num = dequeue(q);

enqueue(q, num);

ptr = q->head;

}

}

**Main.c - לבדיקת הפונקציות**

#include <stdio.h>

#include "Stack.h"

#include "Queue.h"

void main()

{

printf("\n############################### Testing part 2 - Queue ###############################\n");

// Testing queue funtions

printf("\n==============Testing queue functions==============\n");

printf("\n----1st test(initQueue and enqueue and printList\_Queue)----\n");

Queue Test2;

initQueue(&Test2); //test initQueue

enqueue(&Test2, 1); //test enqueue

enqueue(&Test2, 2);

enqueue(&Test2, 3);

enqueue(&Test2, 4);

enqueue(&Test2, 5);

enqueue(&Test2, 6);

enqueue(&Test2, 7);

printList\_Queue(Test2.head); //test printList\_Queue

/\*

printf("\n----next test(dequeue)----\n");

unsigned int num = dequeue(&Test2); //test dequeue

printList\_Queue(Test2.head);

printf("\nThe number that was dequeued is: %d\n", num);

printf("\n----next test(isEmptyQueue and destroyQueue)----\n");

printf("Before destroy: %d\n", isEmptyQueue(&Test2)); //test isEmptyQueue

destroyQueue(&Test2); //test destroyQueue

printf("After destroy: %d\n", isEmptyStack(&Test2)); //test isEmptyQueue

\*/

printf("\n===================================================\n");

// Testing rotateQueue

printf("\n==============Testing rotateQueue==============\n");

printf("\n----1st test(In case queue does not exist)----\n");

rotateQueue(NULL); //In case queue does not exist

printf("\n----next test(Check if rotateQueue worked)----\n");

rotateQueue(&Test2);

printList\_Queue(Test2.head);

printf("\n===================================================\n");

// Testing cutAndReplace

printf("\n==============Testing cutAndReplace==============\n");

printf("\n----1st test(In case queue does not exist)----\n");

cutAndReplace(NULL); //In case queue does not exist

printf("\n----next test(Check if cutAndReplace worked)----\n");

cutAndReplace(&Test2);

printList\_Queue(Test2.head);

printf("\n===================================================\n");

// Testing sortKidsFirst

printf("\n==============Testing sortKidsFirst (includes minValueIndex and moveIndexToHead)==============\n");

printf("\n----1st test(In case queue does not exist)----\n");

sortKidsFirst(NULL); //In case queue does not exist

printf("\n----next test(Check if sortKidsFirst worked)----\n");

sortKidsFirst(&Test2);

printList\_Queue(Test2.head);

printf("\n===================================================\n");

}

**פלט:**