#include <stdio.h>

typedef struct node

{

int data;

struct node \*next1;

struct node \*next2;

}Node;

Node \*createNode(int data)//הוספת צומת לרשימה דו כיוונית

{

Node \*temp = (Node \*)malloc(sizeof(Node));

temp->data = data;

temp->next1 = NULL;

temp->next2 = NULL;

}

Node \*addToFirst(Node \*head, int data)//הוספת צומת לתחילת הרשימה

{

Node \*temp = createNode(data);

temp->next1 = head;

if (head!=NULL)

head->next2 = temp;

return temp;

}

Node \*addToLast(Node \*head, int data)//הוספת צומת לסוף הרשימה

{

Node \*temp = createNode(data);

Node \*p = head;

if (head == NULL)

return temp;

while (p->next1 != NULL)

p = p->next1;

p->next1 = temp;

temp->next2 = p;

return head;

}

int length(const node\_t \*head) /\*פונקצה לבדיקה אורך הרשימה\*/

{

if (head == NULL)

return 0;

return 1 + length(head->next);

}

node\_t\* find(node\_t \*head, int val) /\*חיפוש ערך ברשימה המקושרתנ

הפונקציה תקבל את ראש הרשימה ואת הערך שהיא מחפשת ברשימה\*/

{

if (head == NULL)

return NULL;

return (head->data == val) ?

head :

find(head->next, val);

}

Node \*func(Node \*head)/\*הפונקציה מוצאת את האיבר הכי נמוך ברשימה ומעבירה אותו לראש הרשימה\*/

{

Node \*curr = head, \*first = head, \*a\_curr, \*b\_curr, \*b\_temp;

int flag = 0;

int min = curr->num; /\*הגדרת האיבר הראשון להשוואה\*/

while (curr != NULL)

{

if (min > curr->num) /\*ביצוע השוואה אל מול המינימום עד עכשיו\*/

{

min = curr->num; /\*שמירה של המינימום החדש\*/

first = curr; /\*שמירת הכתובת של תא המינימום החדש\*/

b\_curr = b\_temp; /\*השמת כתובת התא הקודם\*/

a\_curr = curr->next; /\*השמת כתובת התא הבא\*/

flag = 1; /\*הדלקת מפסק\*/

}

b\_temp = curr; /\*שמירת הכתובת של התא הקודם\*/

curr = curr->next;

}

if (flag == 1) /\*במידה והמפסק דלוק\*/

{

first->next = head; /\*השמת ראש הרשימה המקורית במצביע של תא המינימום\*/

head = first; /\*קבעית ראש רשימה חדש\*/

b\_curr->next = a\_curr; /\*ביצוע חיבור של התאים לפני ואחרי תא המינימום\*/

}

return (head);

}

node\_t \*delete(node\_t \*head, int value)//\*מחיקה צומת ברשימה הפונקציה מקבלת את ראש הרשימה ואת הערך שנרצה למחוק ברשימה

{

node\_t \*iter = head, prev = NULL;

if (head == NULL)

return head;

if (head->data == value)

{

iter = head->next;

free(head);

return iter;

}

while (iter->next != NULL)

{

if (iter->data == value)

{

prev->next = iter->next;

free(iter);

break;

}

prev = iter;

iter = iter->next;

}

return head;

{

void Print\_number\_Node( Node\* head)//מדפיסה את המספרים במקושרת

{

Node\* temp = head;

while (temp != NULL)

{

printf(" %d ", temp->num);

temp = temp->next;

}

}

void Print\_addres\_Node(Node\* head)//מדפיסה את הכתובות של המקושרת

{

temp = head;

printf("\n");

while (temp != NULL)

{

printf("%p ", temp);

temp = temp->next;

}

}

Node \* reverseLinkedList(Node \* head) //הופך את הרשימה מקושרת

{

Link \* p = NULL;

while (head != NULL)

{

Node \*q = head;

head = head->next;

q->next = p;

p = q;

}

return p;

}

int palindrom(Node \*head)// הפונקציה בודקת ומחזירה 1 אם הרשימה המקושרת היא פולינדרום

{

Node \*start = head;

Node \*end;

int i;

int count = 1;

while (end->next1 != NULL)//מעביר את המצביע לסוף

{

count++;

end = end->next1;

}

for (i = 0; i < count / 2; i++)// בודק עד חצי הרשימה כי אין צורך לסרוק עד הסוף

{

if (start->n != end->n)

return 0;

start = start->next1;

end = end->next2;

}

return 1;

}

void reverse(Node \*head)//הפונקציה הופכת את הרשימה המקושרת

{

Node \*start = head;

Node \*end;

int i;

int temp;

int count = 1;

while (end->next1 != NULL)//מעביר את המצביע לסוף

{

count++;

end = end->next1;

}

for (i = 0; i < count / 2; i++)

{

//עושה החלפה

temp = start->n;

start->n = end->n;

end->n = temp;

//

start = start->next1;//מקדם את התחלה

end = end->next2;//מקדם אחורה

}

}

int checkSequence(Node \*head)// הפונקציה בודקת ומחזירה 1 אם הרשימה המקושרת היא סידרה חשבונית

{

Node \*start = head;

int diff;//ההפרש

if (head == NULL || head->next1 == NULL)

return 1;

diff = head->next1->n - (head->n);// שם את הפרש הראשון במשתנה לבדיקה

start = start->next1;

while (start->next1 != NULL)

{

if (start->n + diff != start->next1->n)// בדיקה בדרך השלילה אם הפרש שונה בין מספר למספר היא תחזיר 0

return 0;

start = start->next1;

}

return 1;

}

void sort\_f(Node\*head)//הפונקציה ממינת את הרשימה מהקטן לגדול

{

int temp;

Node \*cur, \*min, \*index;

for (cur = head;; cur != NULL; cur = cur->next)

{

index = cur;

min = cur;

for (temp = (\*cur).num; min != NULL; min = min->next)

{

if ((\*min).num < temp)

{

index = min;

temp = (\*min).num;

}

}

(\*index).num = (\*cur).num;

(\*cur).num = temp;

}

}

Node \*sort\_f(Node \*head)//פונקציה לסידור רשימה מקושרת עם כתובות כל צומת תצביע לצומת הבאה בסדר מהקטן לגדול

{

Node \*a\_cur, \*b\_cur, \*cur;

int i = 0, temp = 0, flag = 1;

cur = head;

a\_cur = cur->next;

b\_cur = cur;

while (cur != NULL)

{

cur = cur->next;

i++;

}

temp = i;

cur = head;

for (; a\_cur != NULL; cur = cur->next, a\_cur = a\_cur->next)

{

if (temp != i)

{

cur = head;

a\_cur = cur->next;

b\_cur = cur;

flag = 1;

temp = i;

}

if ((\*cur).num > (\*a\_cur).num)

{

temp--;

if (flag == 1 && cur == head)

{

flag = 0;

cur->next = a\_cur->next;

a\_cur->next = cur;

head = a\_cur;

}

else {

b\_cur->next = a\_cur;

cur->next = a\_cur->next;

a\_cur->next = cur;

}

}

b\_cur = cur;

}

return head;

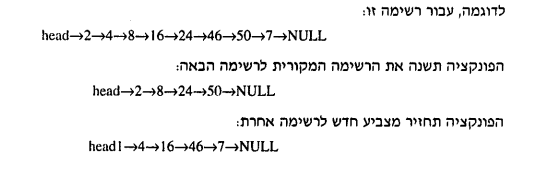
}

**הפונקציה תשנה את הרשימה שהתקבלה כך שרק המשתנים במקומות האי זוגי היו ברשימה**

**(האיבר הראשון הוא במיקום 1 כלומר אי זוגי )**

**ואת האברים במיקום הזוגי תשרשר לרשימה אחרת לפי הסדר שבו הם מופיעים ברשימה המקורית**

**הפונקציה תחזיר מצביע לרשימה החדשה\*/**

****

typedef struct node {

int num;

struct node \*next;

}Node;

Node \*func(Node \*head)

{

Node \*curr, \*headOdd,\*currOdd,\*currZugi,\*headZugi;

int i, counter = 0, zugi = 0, odd = 0;

for (i = 0, curr = head; curr != NULL; curr = curr->next)

{

i++;

if (i == 1)

{

if ((i % 2) != 0)

{

headOdd = curr;

currOdd= headOdd;

}

}

else

{

if ((i % 2) != 0)

{

currOdd->next = curr;

currOdd = currOdd->next;

}

}

if (i == 2)

{

if((i % 2) == 0)

{

headZugi = curr;

currZugi= headZugi;

}

}

else

{

if ((i % 2) == 0)

{

currZugi->next = curr;

currZugi = currZugi->next;

}

}

}

currOdd->next = NULL;

currZugi->next = NULL;

head=headOdd;

return(headZugi);

}

void main()

{

Node \*head, \*curr,\*new\_head;

int num= 0, counter = 0;

puts("enter the size for the list\n");

scanf("%d", &num);

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

while (1)

{

if (num==counter)

{

break;

}

counter++;

if (counter == 1)

{

curr = (Node\*)malloc(sizeof(Node) \* 1);

scanf("%d", &curr->num);

head = curr;

curr->next = NULL;

}

if (counter > 1)

{ curr->next=(Node\*)malloc(sizeof(Node) \* 1);

curr = curr->next;

scanf("%d", &curr->num);

curr->next = NULL;

}

}

new\_head = func(head);

for (curr = new\_head; curr != NULL; curr = curr->next)

{

printf("%d ", curr->num);

puts("\n");

}

puts("the original list now is :\n");

for (curr = head; curr != NULL; curr = curr->next)

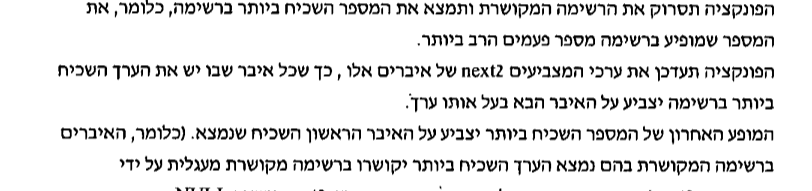
{

printf("%d ", curr->num);

puts("\n");

}

}



#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

int num;

struct node \*next;

struct node \*next2;

}node;

void link\_freq(node \*head)

{

node \*current, \*cur\_2, \*cur\_head;

int flag = 1,i,max=0;

int \*arr = (int\*)malloc(sizeof(int) \* 10);

for (current = head; current != NULL; current = current->next)

{

arr[current->num]++;

if(current->next == NULL)

for (i = 0; i < 10; i++)

{

if (max > arr[i])

{

max = arr[i];

}

}

}

for (current = head; current != NULL; current = current->next)

{

if (current->num == max)

{

if (flag == 1)

{

cur\_2 =cur\_head= current;

flag = 0;

continue;

}

else {

cur\_2->next2 = current;

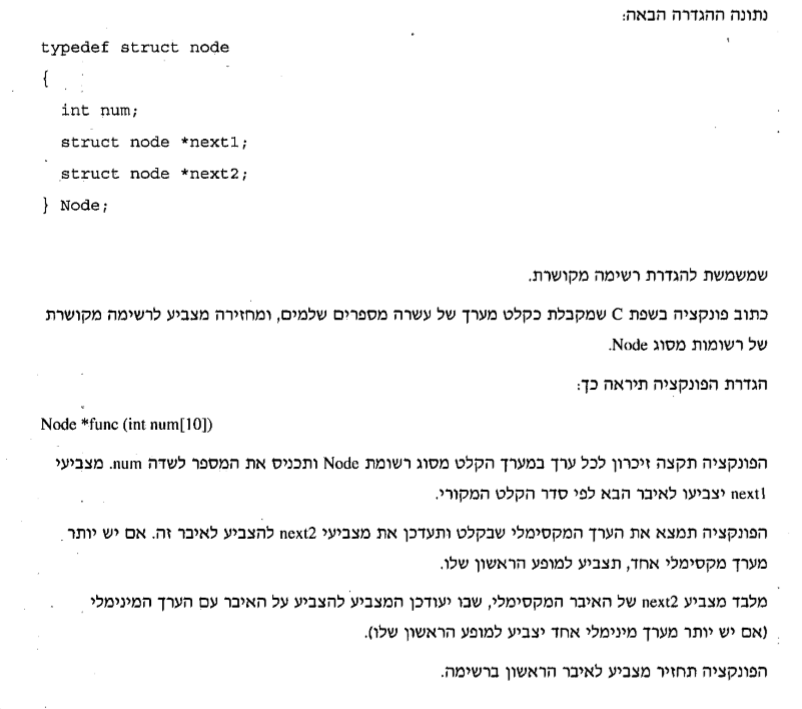
}

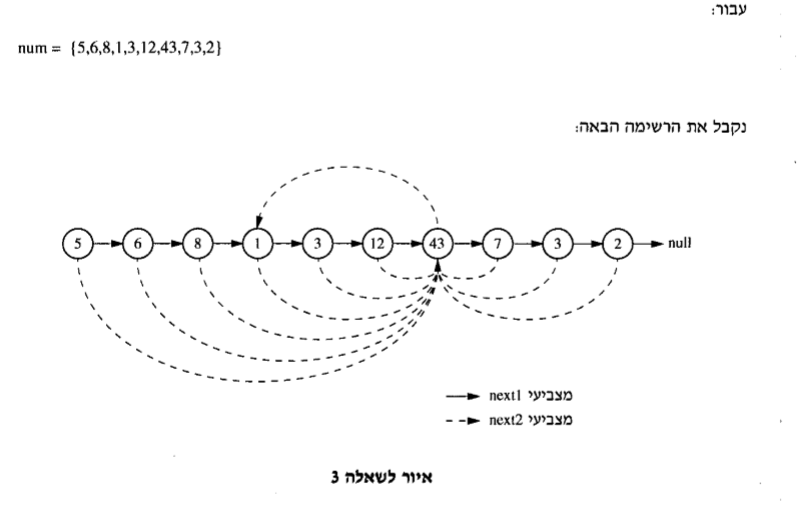
}

}

cur\_2->next2 = cur\_head;

}



#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int num;

struct node \*next1;

struct node \*next2;

}Node;

Node \* createNode(int);

Node \* addToLast(Node \*, int);

Node \*func(int num[10]);

Node \*findMax(Node \*head);

Node \*findMinimum(Node \*head);

void printList(Node \*head);

void main()

{

int a[] = { 1,2,3,4,5,6,7,8,9,10 };

Node \*head = func(a);

//printf("%d", head->next2->num);

printList(head);

}

void printList(Node \*head)

{

Node \*p = head;

while (p != NULL)

{

printf("\n%d", p->num);

printf(" In Address = %p", p);

printf("\n%d", p->next1->num);

printf("\n%d", p->next2->num);

p = p->next1;

}

}

Node \*func(int num[10])

{

Node \*head = NULL;

Node \*p;

Node \*maxNode;

int i;

for (i = 0; i < 10; i++)

head = addToLast(head, num[i]);

maxNode = findMax(head);

p = head;

while (p != NULL)

{

p->next2 = maxNode;

p = p->next1;

}

maxNode->next2 = findMinimum(head);

return head;

}

Node \*createNode(int num)

{

Node \*temp = (Node \*)malloc(sizeof(Node));

temp->num = num;

temp->next1 = NULL;

temp->next2 = NULL;

return temp;

}

Node \*addToLast(Node \*head, int num)

{

Node \*p = head;

Node \*temp = createNode(num);

if (head == NULL)

return temp;

while (p->next1 != NULL)

p = p->next1;

p->next1 = temp;

return head;

}

Node \*findMax(Node \*head)

{

int max = head->num;

Node \*maxNode = head;

Node \*p = head;

while (p != NULL)

{

if (p->num > max)

{

max = p->num;

maxNode = p;

}

p = p->next1;

}

return maxNode;

}

Node \*findMinimum(Node \*head)

{

int minimum = head->num;

Node \*minimumNode = head;

Node \*p = head;

while (p != NULL)

{

if (p->num < minimum)

{

minimum = p->num;

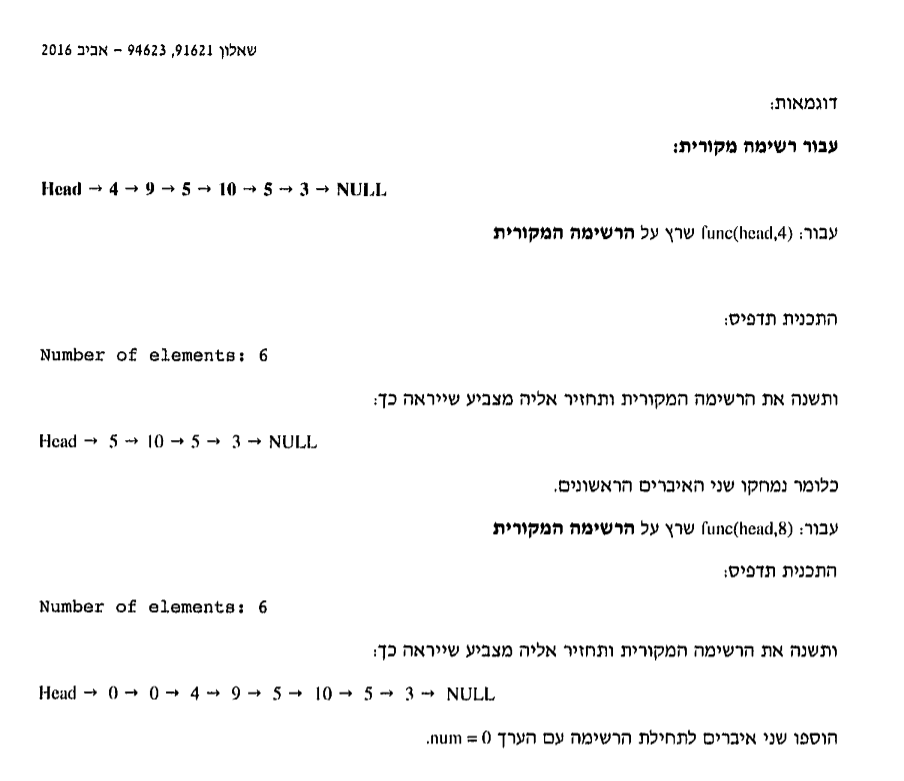
minimumNode = p;

}

p = p->next1;

}

return minimumNode;



#include<stdio.h>

typedef struct node

{

int num;

struct node \*next;

}Node;

Node \*Func(Node \*head, int number)

{

int i = 0, j = 0, flag = 0;

Node \*Point, \*cur,\*headpoint;

cur = head;

while (cur != NULL)

{

i++;

cur = cur->next;

}

printf("%d", i + 1);

flag = number - i;

cur = head;

if (flag > 0)

{

headpoint = (Node\*)malloc(sizeof(Node));

Point = headpoint;

while (flag != 0)

{

flag--;

if (flag)

{

Point->next = (Node\*)malloc(sizeof(Node));

}

else{Point->next = head;

Point->num = 0;

head = headpoint;

break;

}

Point->num = 0;

Point = Point->next;

}

}

else {

if (flag < 0)

{

while (flag != 0)

{

Point = head;

head = head->next;

Point->next = NULL;

free(Point);

flag++;

}

}

}

return head;

}

void main()

{

Node \*head, \*cur;

head = (Node\*)malloc(sizeof(Node));

int number = 4,i=0;

for (cur = head; cur != NULL; cur = cur->next)

{

cur->num =i+++1;

if (i == 7)

{

cur->next = NULL;

}

else {

cur->next= (Node\*)malloc(sizeof(Node));

}

}

head=Func(head, number);

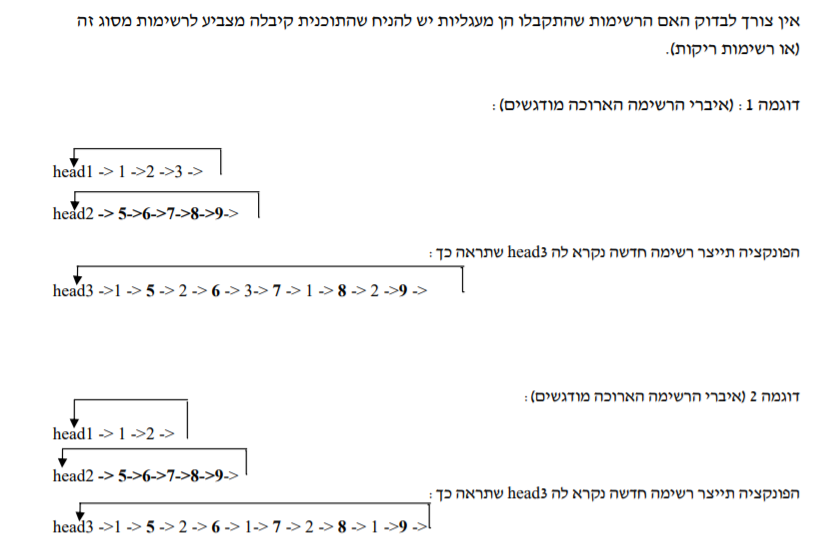
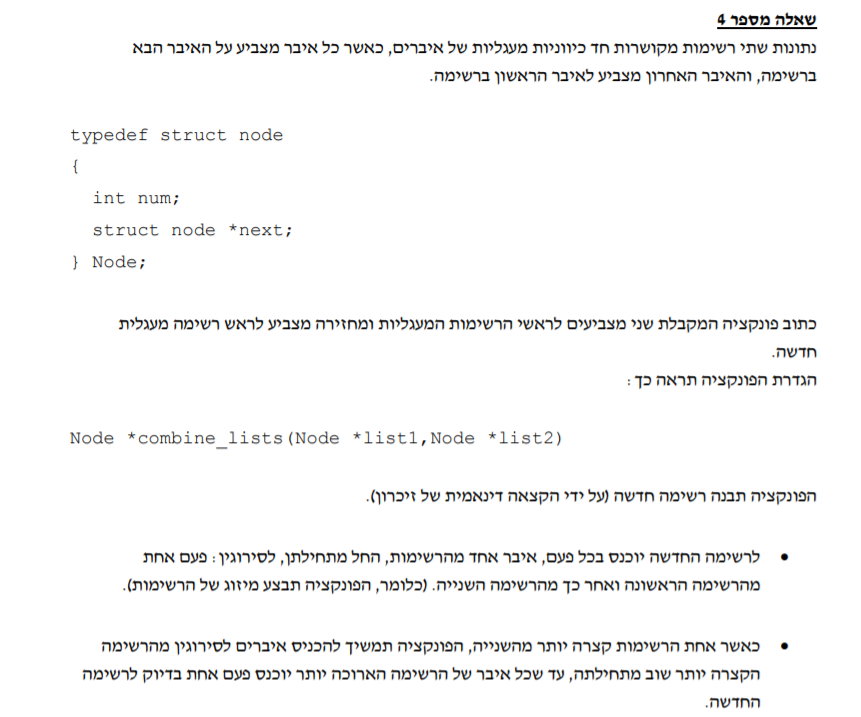
for (cur = head; cur != NULL; cur = cur->next)

{

printf("%d", cur->num);

}

}



#include<stdio.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int num;

struct node \*next;

} Node;

Node \*input(int size); //פונקציה לאתחול הרשימות

Node \*combine\_lists(Node \*list1, Node \*list2); //פונקציה לאיחוד הרשימות

int main()

{

Node \*list1, \*list2, \*new\_list;

int size = 0;

printf("Enter size for list 1: ");

scanf("%d", &size);

if (size == 0)

list1 = NULL;

else if (size>0)

list1 = input(size);

printf("Enter size for list 2: ");

scanf("%d", &size);

if (size == 0)

list2 = NULL;

else if (size>0)

list2 = input(size);

new\_list = combine\_lists(list1, list2);

if (new\_list == NULL)

printf("inValid input.\n");

else if (new\_list)

{

printf("New combined list: ");

while (new\_list)

{

printf("%d->", new\_list->num);

new\_list = new\_list->next;

}

puts("");

}

return (0);

}

Node \*input(int size)

{

Node \*head, \*curr, \*new\_link;

int i;

for (i = 0; i < size; i++)

{

if (!i)

{

head = (Node\*)calloc(1, sizeof(Node));

curr = head;

printf("Enter num[%d]: ", i + 1);

scanf("%d", &curr->num);

}

else if (i)

{

new\_link = (Node\*)calloc(1, sizeof(Node));

curr->next = new\_link;

curr = curr->next;

printf("Enter num[%d]: ", i + 1);

scanf("%d", &curr->num);

}

}

curr->next = NULL;

return (head);

}

Node \*combine\_lists(Node \*list1, Node \*list2)

{

Node \*new\_list, \*curr\_short, \*curr\_long, \*before, \*new\_link, \*aux;

int flag = 0, size1 = 0, size2 = 0;

if (list1 == NULL&&list2 == NULL) //במידה ושתי הרשימות ריקות

return (NULL);

if (list1 == NULL || list2 == NULL) //במידה ורק רשימה אחת ריקה

{

if (list1 == NULL)

{

new\_list = list2;

return (new\_list);

}

else if (list2 == NULL)

{

new\_list = list1;

return (new\_list);

}

}

else if (list1&&list2) //מציאת גודל הרשימות

{

aux = list1;

while (aux)

{

size1++;

aux = aux->next;

}

aux = list2;

while (aux)

{

size2++;

aux = aux->next;

}

if (size1<size2) //חלוקת המצביעים על פי אורכי הרשימות

{

curr\_short = aux = list1;

curr\_long = list2;

}

else if (size1>size2) //כנ"ל

{

curr\_short = aux = list2;

curr\_long = list1;

}

}

while (curr\_short&&flag != 2)

{

if (flag == 0) //קביעת ראש הרשימה המאוחדת

{

new\_list = (Node\*)calloc(1, sizeof(Node));

new\_list->num = curr\_short->num;

before = new\_list;

flag = 1;

}

else if (flag == 1) //מהפעם השנייה והילך - הוספה רגילה

{

new\_link = (Node\*)calloc(1, sizeof(Node));

new\_link->num = curr\_short->num;

before->next = new\_link;

before = new\_link;

}

new\_link = (Node\*)calloc(1, sizeof(Node)); //יצירת הקצאה עבור הוספת ערך מהרשימה הארוכה

new\_link->num = curr\_long->num;

before->next = new\_link;

before = new\_link;

if (curr\_long->next == NULL) //במידה והגענו לסוף הרשימה הארוכה הנפת דגל ויציאה

flag = 2;

else

curr\_long = curr\_long->next;

if (curr\_short->next == NULL) //במידה והגענו לסוף הרשימה הקצרה החזרת המצביע לראש הרשימה

curr\_short = aux;

else

curr\_short = curr\_short->next;

}

return (new\_list);

}