**מבנים:**  **תאריך: 4.1.18**

**סטודנטים יקרים, המסמך בא לעזור לכם בהבנת כתיבת רשימות מקושרות, אני כאן כתמיד לכל שאלה שתהיה לכם – אפי 052-7291300**

**הגדרת מבנה:**

המבנה יכול להיות מוגדר מאיזה משתנה שאתם רוצים.

typedef struct NODE {

int data;

int age;

char name[100];

char ch;

}Node;

בתוכנית הבאה קלטנו את המחרוזת effi לתוך המשתנה name שמוגדר בתוך המבנה student והדפסנו אותו.

int main() {

Node Student;

int i = 0, id;

char ch[] = "effi";

strcpy(Student.name , ch);

puts(Student.name);

getchar();

getchar();

return 0;

}

בתוכנית הבאה הגדרנו מערך בגודל 4 מטיפוס node וקלטנו אליו ערכים, לאחר מכן הדפסנו את הערך data שנמצא בתוך האיבר ה 3 במערך.

int main() {

Node Student[4];

int i = 0, id;

char ch;

for (i = 0; i < 4; i++)

{

scanf("%d", &id);

Student[i].data = id;

scanf("%c", &ch);

Student[i].ch = ch;

}

printf("%d", Student[3].data);

getchar();

getchar();

return 0;

}

בתרגילים הבאים נשתמש במבנה הבא:

typedef struct NODE {

int data;

struct NODE \*next;

} Node;

בפונקציה הבאה ניצור איבר חדש מסוג node ונכניס אליו ערכים שקיבלנו מה main ( ניתן לקבל את הערכים גם דרך המשתמש ).

Node \*makeNode(int key)

{

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

if (!newNode)

return NULL;

newNode->data = key;

newNode->next = NULL;

return newNode;

}

בפונקציה הבאה אנחנו נקלוט איברים לסוף הרשימה המקושרת הקיימת – האיבר החדש יתווסף לסוף הרשימה

//adds a new item to the end of the list

Node \* add\_last(Node \*head, Node \* newItem)

{

Node \*currItem;

currItem = head;

while (currItem->next != NULL) //= while (currItem->next)

currItem = currItem->next;

currItem->next = newItem;

return head;

}

בפונקציה הבאה אנחנו נוסיף איבר לאמצע הרשימה המקושרת – האיבר החדש יתווסף למיקום שבו הוא יהיה קטן מהאיבר מימינו ויותר גדול מהאיבר משמאלו

Node \*add\_middle(Node \*head, Node \* newItem)

{

Node \*currItem;

if (!head)

return newItem;

currItem = head;

while (currItem->next) {

if (currItem->data < newItem->data && currItem->next->data > newItem->data)

{

newItem->next = currItem->next;

currItem->next = newItem;

break;

}

currItem = currItem->next;

}

return head;

}

הפונקציה הבאה מפצלת את הרשימה כך שתוחזר רשימה חדשה של כל האיברים הזוגיים ברשימה המקורית

Node \*splitListToEven(Node \*head, Node \*lstEven)

{

Node \*currItem;

Node \*currItemeven = NULL;

if (!head)

return NULL;

currItem = head;

while (currItem) {

if (currItem->data % 2 == 0)

{

if (lstEven == NULL)

{

lstEven = makeNode(currItem->data);

currItemeven = lstEven;

}

else

{

currItemeven->next = makeNode(currItem->data);

currItemeven = currItemeven->next;

}

}

currItem = currItem->next;

}

return lstEven;

}

הפונקציה הבאה מפצלת את הרשימה כך שתוחזר רשימה חדשה של כל האיברים האי זוגיים ברשימה המקורית

Node \*splitListToOdd(Node \*head, Node \*lstOdd)

{

Node \*currItem;

Node \*currItemodd = NULL;

if (!head)

return NULL;

currItem = head;

while (currItem) {

if (currItem->data % 2 == 1)

{

if (lstOdd == NULL)

{

lstOdd = makeNode(currItem->data);

currItemodd = lstOdd;

}

else

{

currItemodd->next = makeNode(currItem->data);;

currItemodd = currItemodd->next;

}

}

currItem = currItem->next;

}

return lstOdd;

}

בקטע הבא נראה הדפסה של כל האיברים בתוך רשימה מקושרת:

Node \*temp = head;

while (temp) {

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

temp = temp->next;

}