תרגיל בית 3 (שפת C)

- בתרגיל זה **אין להשתמש בהקצאות זיכרון דינמיות** (קראו את כל התרגיל ותבינו איך זה אפשרי)
- בתרגיל זה אין להשתמש ב inline functions
- מותר להשתמש בפונקציות ב stdlib.h

הנחיות כלליות לפיתרון התרגילים בקורס:

- התרגילים הם לעבודה ביחידים. **מותר להתייעץ אך ורק בעל פה, אסור בתכלית האיסור שחומר כתוב/מודפס/אלקטרוני יעבור בין אנשים. בנוסף, על חלק מהתרגילים תיבחנו פרונטלית ועליכם להבין כל דבר בקוד!**
- המנעו ממספרי קסם: מספרים שמופיעים באמצע הקוד בלי משמעות מיוחדת (לדוגמא נניח שמספר הרשומות בתרגיל אחר הוא מקסימום 50 ואז בכל מקום בקוד כתוב 50. לעומת זאת, 0 לתחילת מערך לא נחשב מספר קסם - הפעילו הגיון בריא) והשתמשו במקום זאת בפקודות מאקרו (#define או אם כבר למדתם על כך ב const).
- אין להשתמש ב-variable length arrays , וכן במשתנים סטאטיים ו / או גלובליים.
- כל התרגילים בקורס צריכים להתקמפל ולרוץ באתר c9.io (האתר מריץ מערכת הפעלה אובונטו) עם שורת הקמפול:

  gcc -Wall -Wvla -Werror -g ...

  עבור תרגילי C ושורת הקמפול:

  g++ -Wall -Wvla -Werror -g -D_GLIBCXX_DEBUG -std=c++11 ...

  עבור תרגילי C++
- או במידה ומצורף Makefile עם ה Makefile המצורף
- יש להקפיד על סגנון תכנות טוב כמו שלמדתם. לדוגמא, להימנע מחזרות קוד (לכתוב פונקציות שצריך), שמות משתנים עם משמעות, בהירות הקוד, תיעוד הקוד, להקפיד להשתמש בקבועים שצריך ולא במספרי קסם וכו'.
- <u>אופן כתיבת ההערות</u>: בכל תחילת קובץ הצהרות (.h או .hpp) יש לכתוב את תפקיד הקוד שבקובץ.
  כמו-כן לפני כל פונקציה ומתודה ומשתנה מחלקה/מבנה יש לכתוב הערה על תפקידם.
  יש להוסיף הערות במימוש לפי הצורך.
  בקורס זה במידה וניתנו לכם הוראות מפורטות לגבי פונקציה בתרגיל מותר לכתוב ראה בהגדרת התרגיל במקום לעשות copy&paste.
- עליכם להגיש קובץ ששמו מספר ת.ז. שלכם כמו שהיא מופיעה באתר המודל נקודה zip . לדוגמא, אם מספר ת.ז. שלי הוא 12345678 אז שם הקובץ יהיה:

  12345678.zip
- **שאלות על התרגיל יש לשאול בפורום המתאים במודל בלבד!**

<u>שאלה 1</u>

בשאלה זו נתרגל קריאה וכתיבה לקבצים, ושימוש במערכים רב ממדיים.

תמונות דיגיטליות מיוצגות באמצעות טבלת מספרים (או מטריצה) כאשר המספר בכל תא בטבלה מייצג עוצמת אור מסויימת. בדרך זו מייצגים גם אותות ממקורות אחרים כמו גלי מכ"ם, חום, קרינה וכדומה.

מכיון שמדידות פיזיקליות אינן מושלמות, עשויה להתווסף לכל תא במטריצה כמות קטנה (חיובית או שלילית) של רעש באופן אקראי. על מנת לסנן רעש מקובל לבצע טשטוש של האות באמצעות מיצוע ערך כל תא עם ערכי התאים המצויים בסביבתו הקרובה, על בסיס ההנחה לפיה הערכים של תאים קרובים אמורים להיות בדרך כלל דומים.

עבור כל תא ij במטריצת הקלט נחשב את ממוצע הערכים של תת מטריצה ריבועית או 'חלון' בגודל (w*w) שהתא ממוקם במרכזו, ונשמור את התוצאה במטריצה אחרת במקום ה-ij.

ניתן לממש חישוב זה באופן נאיבי על ידי ביצוע מספר פעולות פרופורציוני לגודל החלון כפול מספר התאים במטריצה. בתרגיל זה **אין לממש באופן הנאיבי.** בתרגיל זה תממשו דרך אלגנטית יותר המאפשרת לחשב את ממוצע הערכים עבור כל גודל חלון w, כך שזמן הריצה יהיה תלוי במספר התאים במטריצה **בלבד.**

* *אלגוריתם*

בהינתן מטריצת קלט A , ורדיוס r (מספר שלם), גודל החלון הוא w=2r+1 (מספר שלם ואי זוגי):

1. בצע סכימה מצטברת של כל שורה ב-A באמצעות הוספת ערך כל תא בשורה לזה שאחריו.
2. בצע סכימה מצטברת של כל עמודה ב-A באמצעות הוספת ערך כל תא בעמודה לזה שבא אחריו.
3. צור מטריצה S בגודל זהה ל-A.
4. חשב את הערך של כל תא ij ב-S על פי הנוסחה הבאה:

$$S(i,j) = A(i+r , j+r) - A(i-r-1 , j+r) - A(i+r , j-r-1) + A(i-r-1 , j-r-1)$$

5. בצע חלוקה של כל תא ב-S במספר הפיקסלים בחלון (w^2).

הסבר:
לאחר הסכימה המצטברת על השורות והעמודות של A, התא ה-ij מכיל את סכום הערכים המקוריים של תת המטריצה שפינתה השמאלית העליונה A11, ופינתה הימנית התחתונה Aij. סכום הערכים המקוריים בחלון סביב התא ה-ij ניתן לחישוב על ידי סכומי ארבע תת מטריצות באופן שמזכיר 'הכלה והדחה'. צעד 4 מבצע את חישוב הסכום, ובצעד 5 מתבצע חילוק בגודל החלון בכדי לקבל את ממוצע הערכים שימו לב שזמן החישוב של כל שלב תלוי רק במספר התאים במטריצה, ולא בגודל החלון w.

מידע נוסף על האלגוריתם ושימושיו ניתן למצוא בקישור
http://en.wikipedia.org/wiki/Summed_area_table

טיפול בגבולות המטריצה: שימו לב שהחישוב בסעיף 4 עלול לצאת מגבולות המטריצה. מימוש לא זהיר עלול לגרום לתוצאות בלתי צפויות או לשגיאות בזמן ריצה.
לשם טיפול נכון יש לנהוג לפי העיקרון הבא. אנו מניחים שמחוץ לגבולות המטריצה A ישנם כביכול אפסים בכל כיוון אפשרי, עד אינסוף.
מכך נובע שהסכום עד לנקודה Aij כאשר i או j שליליים, הוא תמיד אפס. לעומת זאת אם שני האינדקסים i ו-j חיוביים, אלא שאחד מהם (או שניהם) חורגים מן הגודל המקסימלי של המטריצה – הסכום המצטבר נשאר כמו בגבול המטריצה, כי מכאן ואילך התווספו רק אפסים. זאת אומרת שצריך ללכת לתא האחרון במטריצה בכיוון בו חרגנו (למשל אם i גדול ממספר השורות n, נפנה ל-Anj).
שימו לב: האיברים מחוץ למטריצה נחשבים כאיברים לכל דבר ועניין. כלומר, שמחשבים ממוצע הם נחשבים בספירה של מספר האיברים.

ממשו את הפונקציה MeanWindow בקובץ MeanWindow.c המקבלת כקלט שתי מחרוזות - שם של קובץ קלט ושם של קובץ פלט. את הכותרות (header) של פונקציות העזר כתבו בקובץ MeanWindowHelperFunctions.h וממשו אותן גם כן בקובץ MeanWindow.c. בקובץ MeanWindow יש להכליל את MeanWindowHelperFunctions.h.
הפונקציה קוראת מטריצה וגודל חלון מתוך קובץ קלט, מבצעת חישוב של הממוצע סביב כל תא (כלומר ממוצע הערכים בחלון בגודל w*w שהתא במרכזו) וכותבת את התוצאה לקובץ הפלט **באותו פורמט בדיוק.**

הפורמט עבור שני הקבצים הוא:

```
n   m   w
a11  a12  a13  ...  a1m
a21  a22  a23  ...  a2m
...
an1  an2  an3  ...  anm
```

כאשר n מספר השורות במטריצה, m מספר העמודות, w גודל החלון, ו-aij ערך התא ה-ij במטריצה.
המספר המקסימלי של שורות או עמודות במטריצה לא יעלה על 100.
המרווחים בכל שורה (כולל השורה הראשונה) קבועים באמצעות tab, וכל שורה של המטריצה מתחילה בשורה חדשה בקובץ .ראו דוגמת קובץ קלט: MeanWindow.in.1 ודוגמת קובץ פלט מתאים: MeanWindow.out.1

הנחיות לקריאה ולכתיבה של קבצים:

- עליכם לבדוק שהתכנית רצה כראוי עם הפרמטרים של שמות קבצי הקלט / פלט הנדרשים.
- עליכם לבדוק שפתיחת הקובץ לקריאה / כתיבה אכן הצליחה. אם היא לא הצליחה עליכם לדווח שגיאה כרצונכם. אם הצליחה אתם יכולים להניח קלט תקין.
- בצעו מעבר יחיד על הקובץ לצורך קריאת / כתיבת הנתונים.
- כל הדרכים לקריאת קבצים או כתיבה לתוכם יתקבלו.
- מידע על פונקציות שימושיות כגון fscanf ,fprintf ,fgets ,fclose ,fopen, strncpy, sscanf ,strcpy ,atoi ,strtok ,strpbrk, ניתן למצוא באתר:
www.cplusplus.com
או על ידי הקלדת man ב-shell של לינוקס.

הנחייה נוספת

- בנוסף לקובץ MeanWindow.c ולקובץ MeanWindowHelperFunctions.h עליכם להגיש גם קובץ MeanWindowTest.cpp (שימו לב שזה הוא קובץ cpp ולא קובץ c , ראו absdiff.zip באתר כדוגמא) אשר יכיל בדיקה של התרגיל ע"י ספריית Google Unit Test. תזכורת: Unit Testing היא בדיקה של כל יחידות התוכנה. חשוב לבדוק את כל פונקציות העזר שכתבתם בנפרד. בנוסף, אפשר לכתוב בדיקות black box testing ודוגמא אחת של בדיקה כזאת יש בקבצים המצורפים לתרגיל.

שאלה 2

בעיית סיווג עצמים היא אחת הבעיות הידועות בתחום הלמידה החישובית. בשאלה זו נממש את אלגוריתם הסיווג Perceptron.

דוגמא טובה להמחשת הבעיה של סיווג עצמים היא זיהוי של תפוזים רקובים.
חקלאים מעבירים לחברת משקאות אלפי תפוזים על מנת לייצר מהם מיץ. הבעיה היא שחלק מן התפוזים המגיעים אל פס הייצור אינם ראויים לשימוש. החברה מחזיקה אנשים שכל תפקידם הוא לעקוב אחרי פס הייצור, לזהות תפוזים רקובים ולהוציא אותם באופן ידני.
בעל החברה מחליט שהגיע הזמן להתקדם לעידן הטכנולוגי, ולהחליף את האנשים ברובוט שיזהה תפוזים רקובים ויוציא אותם באופן אוטומטי מפס הייצור. הוא פונה אליכם כדי שתפתחו עבורו רובוט מתאים.

הבעיה: נניח שהרובוט מסוגל לקבל קלט של תמונה, משקל התפוז וכדומה.
בהינתן תפוז על הרובוט להחליט האם הוא ראוי לשימוש או לא.

פתרון ע"י למידה: נראה לרובוט סדרה של תפוזים, ונעביר לו מידע בנוגע לכל תפוז – האם הוא ראוי לשימוש או לא. הרובוט ייעזר באלגוריתם למידה שמקבל דוגמאות של תפוזים עם תיוג נכון (טובים / רקובים), מבצע הכללה על פי הדוגמאות שקיבל, ומחזיר פונקציית סיווג המזהה עבור תפוזים חדשים אם הם טובים או רקובים. הפונקציה עלולה כמובן לטעות, אבל בסיכוי גבוה מחזירה את התשובה הנכונה.

נתאר את התהליך באופן פורמלי:

1. נתון **מדגם אימון** – אוסף ווקטורים $x_1 \ldots x_n$, כל אחד מהם במימד d:
$x_i = (x_{i1} \ldots x_{id})$
לכל וקטור xi מדגם האימון מכיל גם תגית $y_i = \pm 1$.
במקרה שלנו כל וקטור מייצג תפוז, וכל קואורדינטה מתקבלת ממדידה כלשהי שבוצעה עליו. התגית היא +1 אם התפוז תקין, או -1 אם הוא רקוב

2. אלגוריתם הלמידה מקבל את מדגם האימון ומייצר על פיו **היפותזה** – פונקציה המקבלת וקטור x ממימד d ומחזירה תגית +1 או -1.

3. לאחר שלב הלמידה יש בידינו היפותזה אותה נפעיל על וקטורים חדשים ממימד d (כלומר מדידות על תפוזים שעוד לא ראינו) ונחזיר את הסיווג המתאים. ככל שאלגוריתם הלמידה יהיה מוצלח יותר, הסיכוי לקבל את הסיווג הנכון בשלב זה יהיה גבוה יותר.

* *הפרדה ליניארית*

נתבונן על כל וקטור בקלט כעל נקודה במרחב d מימדי. לשם פשטות מצורפת דוגמא של מרחב דו מימדי (d = 2):

הנקודות החיוביות מייצגות דוגמאות שסווגו +1, והנקודות השליליות מייצגות נקודות שסווגו -1.

במקרה הדו ממדי ההיפותזה שלנו היא קו ישר העובר דרך ראשית הצירים וחוצה את המרחב לשנים. כל נקודה שתצא מצד אחד של הישר תסווג +1, וכל נקודה שתצא מצידו השני תסווג -1. במקרה הכללי נקבל, במקום ישר, **על-מישור** ממימד d-1 שעובר דרך ראשית הצירים (למשל מישור דו ממדי החוצה את המרחב התלת ממדי ל-2, וכן הלאה). הנקודות מצד אחד של העל-מישור יסווגו +1, והנקודות מצידו

השני יסווגו 1-.

קל להבחין בחיסרון המשמעותי של המודל שלנו: למרבית הקלטים במציאות לא קיימת הפרדה ליניארית. ישנן דרכים לשכלל את ההפרדה ולקבל אלגוריתמים חזקים בהרבה, אך לא נממש אותן במסגרת הקורס.

* *אלגוריתם*

ראשית נשים לב כי ניתן לאפיין כל על-מישור מפריד בכל מרחב d מימדי על ידי הווקטור w המאונך לו.

טענה: יהי w וקטור מאונך לעל-מישור כלשהו, ותהיה x נקודה כלשהי במרחב. אם המכפלה הפנימית $<w,x>$ היא חיובית אזי הנקודה נמצאת בצד אחד של העל מישור המפריד (נקרא לו "הצד החיובי"), אחרת הנקודה נמצאת בצד השני (לו נקרא "הצד השלילי").
הערה: המכפלה פנימית $<w,x>$ שווה פשוט ל:

$$w[0]x[0] + \ldots + w[d]x[d]$$

שכזכור d הוא אורך הווקטורים.

מהטענה נובע שניתן לתייג נקודות כ-1+ או 1- על פי מיקומן ביחס לעל-מישור שנבחר.
מטרתנו כעת לבחור על מישור שיפריד נכונה את מדגם האימון, כלומר שכל הדוגמאות x_i עבורן $y_i = +1$ יהיו מצידו החיובי, וכל הנקודות עבורן $yi = -1$ יהיו מצידו השלילי.

נסמן ב-n את מספר הנקודות במדגם האימון. האלגוריתם פועל באופן הבא:
1. נאתחל $w = 0$ (כזכור w הוא וקטור, הכוונה בסימון היא שכל איבריו הם אפס)
2. לכל $i = 1...n$

2.1 נחשב את סימן המכפלה הפנימית $< w , x_i >$ (שימו לב ש w ו x_i הם וקטורים).
2.2 אם הסימן שחישבנו שווה ל-y_i לא נעשה דבר. אחרת נעדכן $w = w + y_i * x_i$ (שימו לב שעבור הדוגמא הראשונה ממנה אנו לומדים הסימן יוצא תמיד אפס, שזה תמיד שונה מ 1+ או 1- ולכן עבור הדוגמא הראשונה תמיד יתבצע עדכון).
3. נחזיר את w בתור הווקטור המייצג את ההיפותזה שמצאנו.

כעת, הסיווג עבור נקודה חדשה x יתקבל על ידי חישוב סימן התוצאה של המכפלה הפנימית בין הווקטור w שקיבלנו מאלגוריתם הלמידה, לבין הווקטור x המייצג את הנקודה שעלינו לסווג.

* *משימת תכנות*

ממשו את הפונקציה LinSeparator בקובץ LinSeparator.c המקבלת כקלט שתי מחרוזות - שם של קובץ קלט (המכיל את מדגם האימון ונתונים חדשים לתיוג), ושם של קובץ פלט. את ה header של פונקציות העזר כתבו בקובץ LinSeparatorHelperFunctions.h וממשו אותן גם כן בקובץ LinSeparator.c.

מבנה קובץ הקלט:
שורה ראשונה – המימד d של הווקטורים הנתונים.
שורה שניה – מספר הדוגמאות החיוביות במדגם האימון.
שורה שלישית – מספר הדוגמאות השליליות במדגם האימון.
השורות הבאות כתובות בפורמט הבא: $x_1, x_2, x_3, \ldots x_d$ (מספרים מופרדים על ידי פסיקים) כאשר

תחילה מופיעות הדוגמאות החיוביות, לאחר מכן הדוגמאות השליליות, ולבסוף מספר לא ידוע של דוגמאות שברצוננו לתייג. על התכנית להדפיס את התיוג של הדוגמאות (1 או 1-) לקובץ הפלט, כל תגית בשורה חדשה (שימו לב שתוכנית ה main לא מדפיסה את ה w הנלמד אלא אך ורק את התיוג של הדוגמאות).
הקובץ LinSeparator.in.1 מכיל קלט טיפוסי עבור התכנית. הקובץ LinSeparator.out.1 מכיל פלט עבור קובץ הקלט. עבור קובץ LinSeparator.in.1 קובץ הפלט שלכם צריך להיות זהה ל LinSeparator.out.1.

הנחות מפשטות

- כל המספרים בקלט מופרדים על ידי פסיק יחיד (ללא רווחים, טאבים או מפרידים אחרים).
- כל שורה (כולל האחרונה) מסתיימת ב-'\n'.
- אורך שורות הקלט הוא לכל היותר 200 (שימו לב שבכך יש חסם על המימד d בו אנו עובדים).
- אין חסם על מספר שורות הקלט (שימו לב שבהנחיות הנוספות כתוב שאין לשמור את כל המדגם בזיכרון)
- הנתונים בקובץ הקלט חוקיים, כאשר d ומספרי הדוגמאות החיוביות והשליליות הם מטיפוס int, והווקטורים מכילים מספרים מטיפוס double (מספרי double מכילים גם מספרים שלמים ובפרט המספרים השלמים שמופיעים בדוגמאות הקלט).
- בסיווג (שימו לב, בזמן הסיווג ולא בזמן הלמידה!) של נקודה חדשה, אם היא נמצאת על העל-מישור המפריד (כלומר סימן המכפלה הפנימית שלה עם w הוא אפס) היא תתוייג כ1+.
- מדגם האימון מכיל לפחות דוגמה אחת חיובית ודוגמה אחת שלילית, והמימד d הוא לפחות 2.

הנחיות נוספות:

- עליכם להשתמש ב-struct לצורך תיאור נקודה בקלט האימון (מערך + תיוג מתאים). שימו לב, ניתן לפתור את התרגיל ללא struct אך אנו דורשים מכם להשתמש בו.
- מותר להגדיר בתרגיל זה את ה struct במלואו עם השדות בקובץ h.
- וודאו שאתם מעבירים את ה-struct באופן יעיל (לא מעתיקים אותו).
- אל תשמרו בזיכרון נתונים ללא צורך (לדוגמא, אין לשמור את כל מדגם האימון בזיכרון).
- הימנעו משכפול קוד. השתדלו להיעזר בפונקציות ספריה מוכנות במקום להמציא את הגלגל מחדש.
- בנוסף, פעלו לפי **ההנחיות לקריאה ולכתיבה של קבצים** המפורטות בסוף שאלה 1.
- עליכם לבדוק שפתיחת הקובץ לקריאה / כתיבה אכן הצליחה ולתת הודעת שגיאה לבחירתכם ואיפה שאתם רוצים. אם הקובץ נפתח אתם יכולים להניח שהפורמט תקין.
- בנוסף לקובץ LinSeparator.c ולקובץ LinSeperatorHelperFunctions.h עליכם להגיש גם קובץ LinSeparatorTest.cpp (שימו לב שזה הוא קובץ cpp ולא קובץ c, ראו absdiff.zip באתר כדוגמא) אשר יכיל בדיקה של התרגיל ע"י ספריית Google Unit Test.
תזכורת: Unit Testing היא בדיקה של כל יחידות התוכנה. חשוב לבדוק את כל פונקציות העזר שכתבתם בנפרד. בנוסף, אפשר לכתוב בדיקות black box testing ודוגמא אחת של בדיקה כזאת יש בקבצים המצורפים לתרגיל.

הערה כללית: מומלץ מאד להשתמש ב Makefile שניתן לקמפול, הרצה, יצירת קובץ ה zip (ע"י make zipfile) ובדיקה (make checkzipfile) על מנת להימנע מהגשת תרגיל לא תקין.

שאלות תיאורטיות - על שאלות כאלו וכדוגמתן תצטרכו לענות בבחינה הפרונטלית (בנוסף לשאלות על הקוד שהגשתם). שימו לב: אין צורך להגיש תשובות לשאלות הללו

1. מתכנת כתב תוכנית הבונה רשימה מקושרת של סדרת מספרים ומדפיסה אותם (הקוד בעמוד הבא) בבדיקת התוכנה הסתבר כי התוכנית אינה עובדת כמצופה.
   a. הסבר מדוע התוכנית לא עובדת כמצופה.
   b. תקנו את התוכנית כך שתעבוד היטב. עליה להדפיס 1 42 3-

קוד לשאלה 1

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:
4:  typedef struct _Node {
5:    int _data;
6:    struct _Node* _next;
7:  } Node;
8:
9:  void List_insert( Node* list, int data ) {
10:   Node* newNode = (Node*)malloc(sizeof(Node));
11:   newNode->_data = data;
12:   newNode->_next = list;
13:   list = newNode;
14: }
15:
16:
17: int main() {
18:   Node* head = NULL;
19:
20:   List_insert( head ,  1 );
21:   List_insert( head , 42 );
22:   List_insert( head , -3 );
23:
24:   Node* currNode= head;
25:   while (currNode!=NULL) {
26:         printf("%d  ", currNode->_data);
27:         currNode = currNode->_next;
28:   }
29:
30:   // code that frees memory ...
31:
32:   return 0;
33: }
```

2. בהתייחס לקוד הבא:

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include <string.h>
4.  int g;
5.  typedef struct {
6.     char *_name;
7.     int _id;
8.  } Entry;
9.  void initEntryVec (Entry ** vec, int n) {
10.         int i;
11.         for (i = 0; i < n; ++i) {
12.            vec[i] = (Entry *) malloc(sizeof (Entry));
13.            vec[i]->_id = i;
14.            vec[i]->_name = (char*)malloc(6);
15.            const char* name= "name";
16.            strcpy (vec[i]->_name, name);
17.         }
18. }
19. int main() {
20.            g= 3;
21.            int i;
22.            Entry ** vec = (Entry **) malloc(sizeof(Entry*)*5);
23.            initEntryVec (vec, 5);
24.            g= vec[0]->_id;
25.            for (i=0; i<5; ++i) {
26.                    printf ("(%s, %d)\n", vec[i]->_name, vec[i]->_id);
27.            }
28.            return 0;
29. }
```

הטבלה הבאה מכילה **כתובות**. עליכם לציין לכל **כתובת** את המיקום שלה בזיכרון (בהתייחס לשמם ולשלב הריצה כאשר התוכנית סיימה לבצע את מספר השורה שבסוגריים): מחסנית, גלובלי, ערימה דינמית, איזור הקוד, לא מוגדר. לדוגמא הכתובת &i לאחר שורה 21 היא כתובת במחסנית. הניחו שכל ההקצאות הדינמיות מצליחות.

כתובת	מספר שורה	מיקום בזיכרון
&g	20	
&i	21	מחסנית

&vec	22	
vec	22	
vec+2	22	
*vec	22	
&vec	9	
*vec	12	
&(vec[i]->_id)	13	
name	15	
vec[i]->_name	16	
&g	24	

בהצלחה!