**מצבעים**

זיכרון המחשב בנוי מבתים – כל בית יש כתובת בזיכרון, משתנים תופסים מקום בזיכרון

int=4 , char=1 וכו'

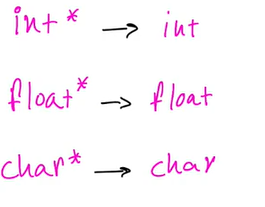
כל משתנה מוקצה בזיכרון – ויש לו כתובת בזיכרון

כל קריאה וכתיבה צריך את הכתובת בזיכרון

מה זה מצביע? זה משתנה שמכיל כתובת של משתנה אחר – חייב להיות מאותו טיפוס

ההבדל ביניהם שמצביע מכיל כתובת של משתנה אחר

כל אחד יש לו את הטיפוס התואם לו:



שני אופרטורים לניהול כתובות

אופרטור \* ש לו שני תפקידים:

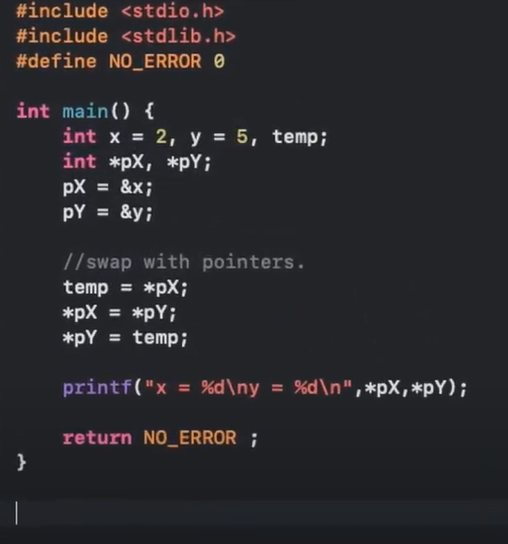
1. יצירת מצביע
2. אחרי היצירה, גישה לכתובת שאליו המצביע מצביע

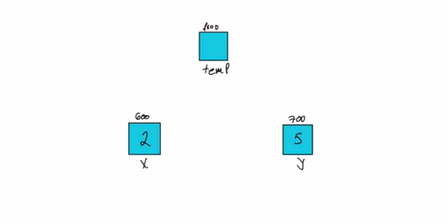
אופרטור & - משתמשים בו (גם &&) או שמשתמשים בו ב scanf קולטים ערכים למשתנים

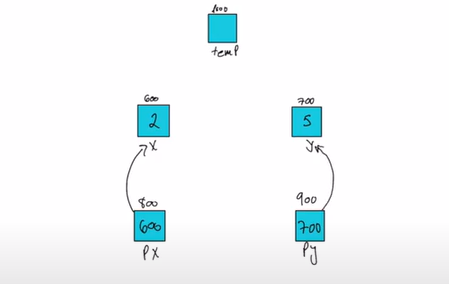
הוא מספק לנו את כתובת של המשתנה

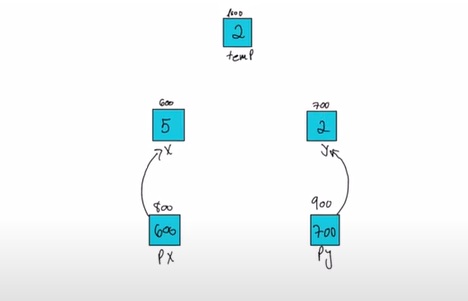
מספר דוגמאות:

החלפת ערכים בין שני משתנים







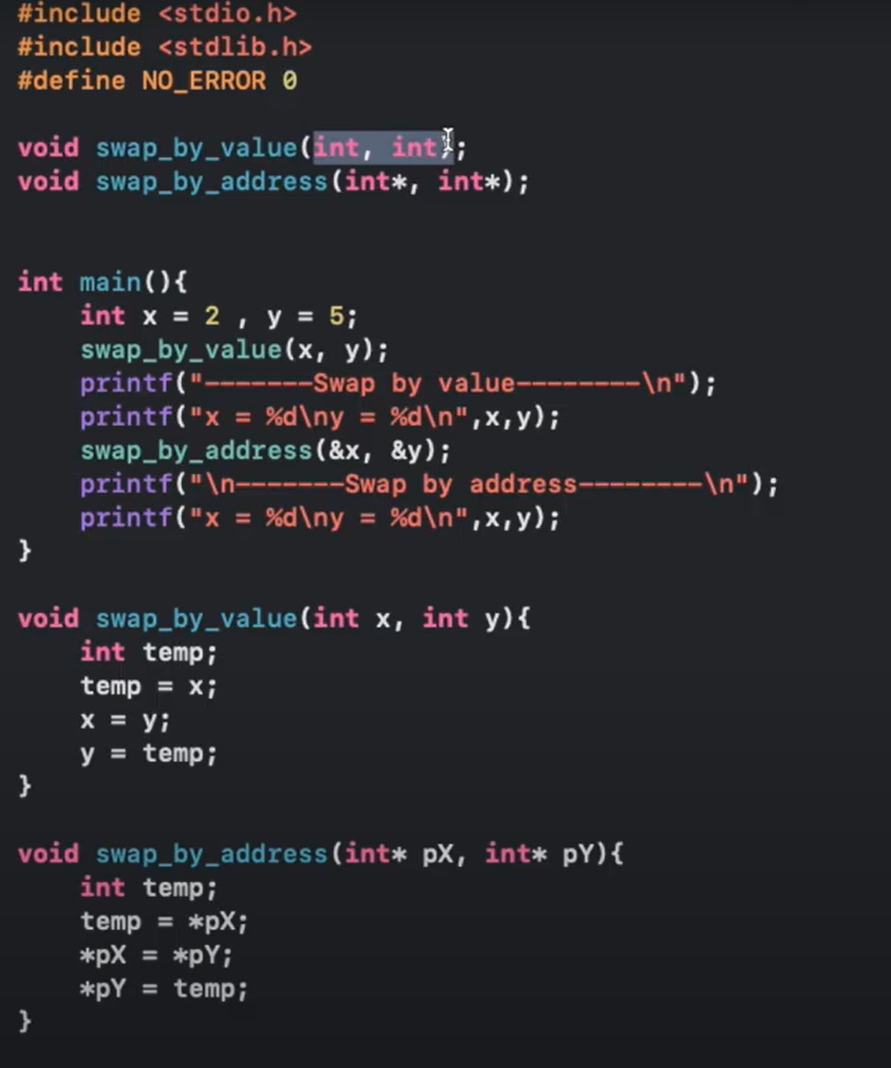


שליחת ערכים by value or by address

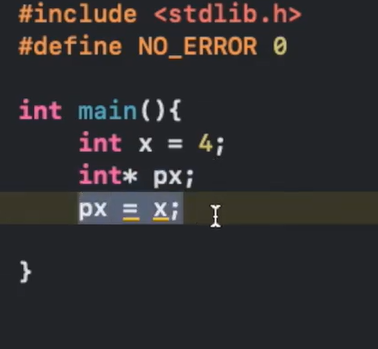
by value - הערך שנשלח מעותק לפרמטר המקומי של הפונקציה

by addres - הכתובת של המשתנה מעובר למשתנה מקומי מסוג מצביע שיאחסן את הכתובת שנשלח ולכן נשתמש ב \* נוכל לגשת לערך עצמו!

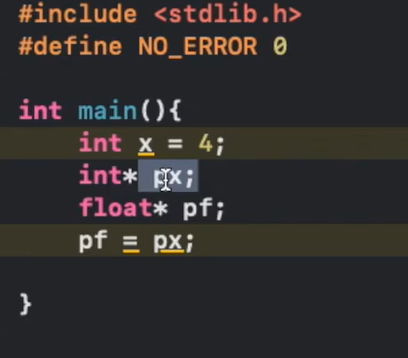
שתי הפוקנציות



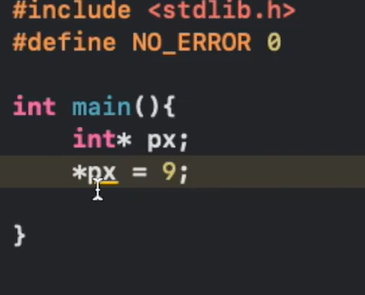
- נפתח זיכרון חדש במחסנית – עבור המשתנים by value



שגיאת קומפלציה – לא ניתן לעשות המרה מ int ל int \*



שגיאת זמן ריצה – כי לא ניתן להמיר int \* ל float\*



שגיאת זמן ריצה - אם ב px אין כתובת של משתנה ( יש בו ערך זבל) אז כאשר עושים \*px מנסים לשים מידע במקום לא מוגדר

המוטיבציה – החזרת ערכים מפונקציה

אנחנו יודעים שניתן להחזיר רק ערך אחד, ע"י מצבעים נוכל להחזיר מספר ערכים

בנוסף, עברת מערכים לפונקציה

הקצאת ערכים בזמן לא ידוע לא ידועים בזמן קופלציה –

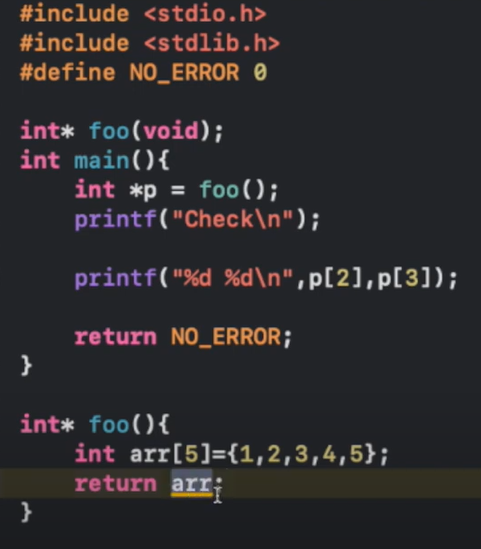
ערך NULL כמו ערך אפס (**אין ערך**) (כמו (\0 מסמן NULL

מצביע על ערך לא חוקי NULL אחרת הוא מצביע לערך חוקי

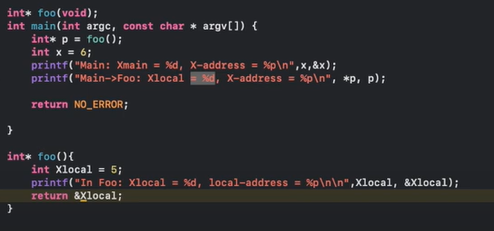
אם מופיע NULL וננסה לגשת אליו יהיה שגיאת זמן ריצה – לכן נבדוק להיכן המצביע מצביע שלא NULL

נהוג לאתחל כל מצביע בערך תקני או NULL

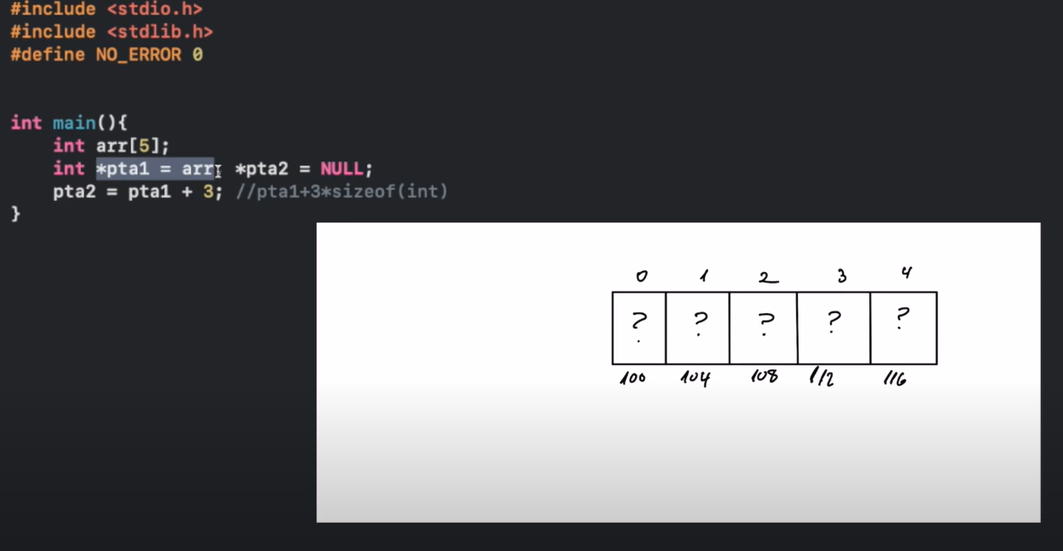
החזרת כתובת של מערך מקומי –

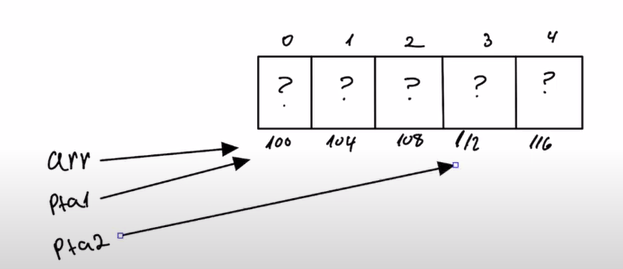


הערכים ב p[2] ו p[3] - ערכים לא מוגדרים - פונקציה נגמרת אז כל הערכים נמחקים מהזיכרון

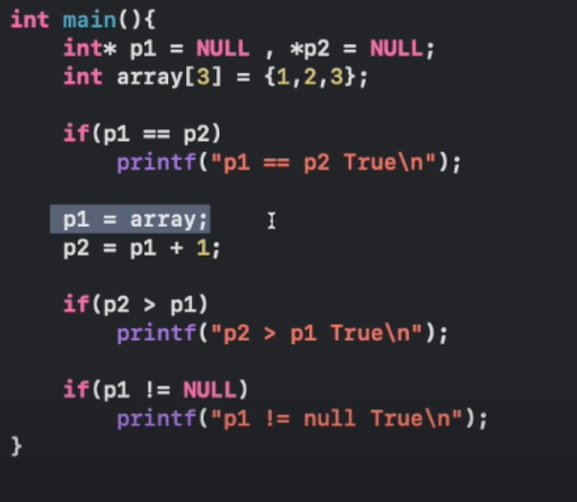


לכן אף פעם לא נחזיר משתנים מקומיים by address !!!





דוגמא נוספת



מערך הוא סוג של מצביע – כי השם של המערך הוא מצביע על הכתובת הראשונה של המערך אבל השם של המערך הוא טיפוס מסוג מצביע קבוע

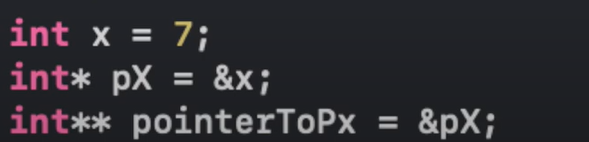
מצביע קבוע – משתנה קבוע לא ניתן לשנות את ערכו ולא ניתן לשנות את הכתובת שאליו הוא מצביע

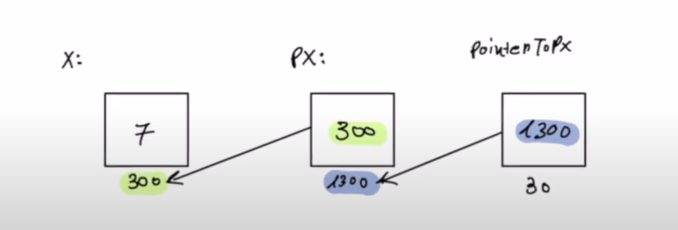
לא יכול שפתאום יצביע על תא שני או שלישי אפשר לגשת לתאים ולשנות אותם אבל לא את הצבעה

כלומר תמיד יצביע על התא הראשון ולא יהיה ניתן לשנות אותו

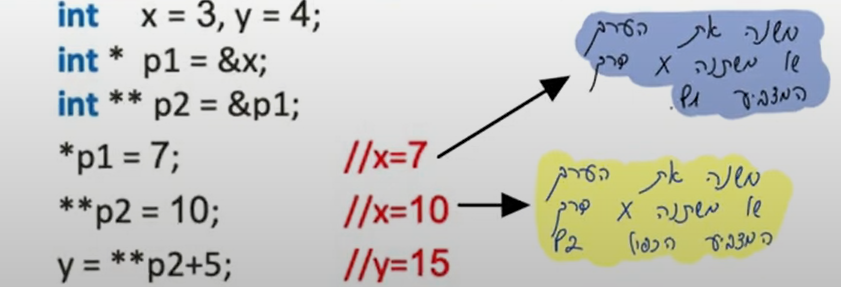
מצביע למצביע

מצביע גם הוא משתנה וגם לו יש כתובת כמו כל משתנה אז נוכל להגדיר מצביע שיצבע למצביע





עוד דוגמא



מערך מצביעים

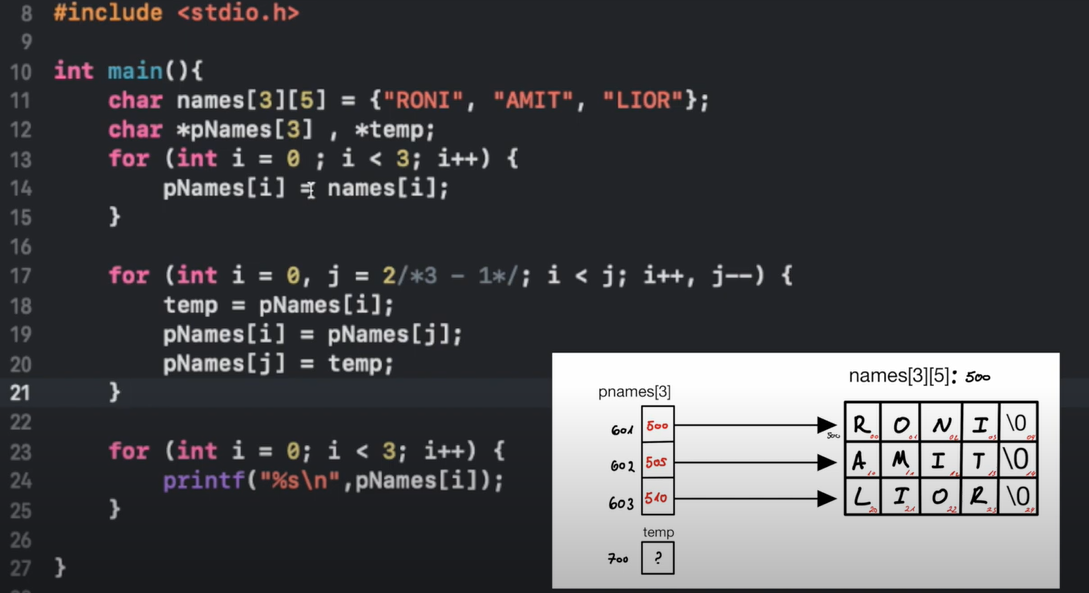
במקום להגדיר מצביע יחיד – מערך שלם של מצביעים

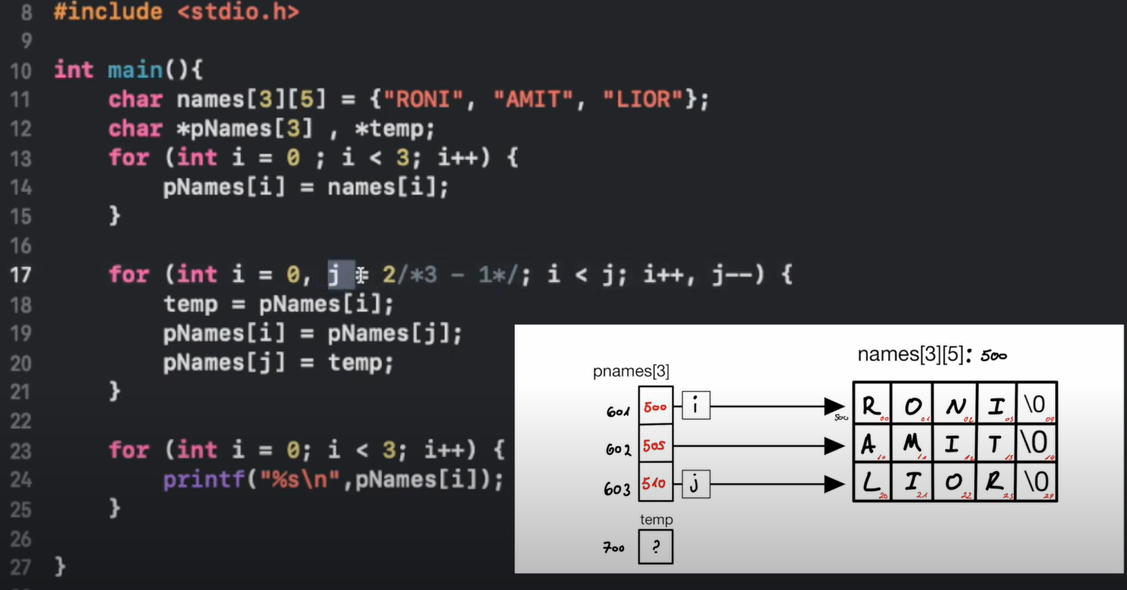
המערך - כל תא במערך מצביע למשתנה מסוים

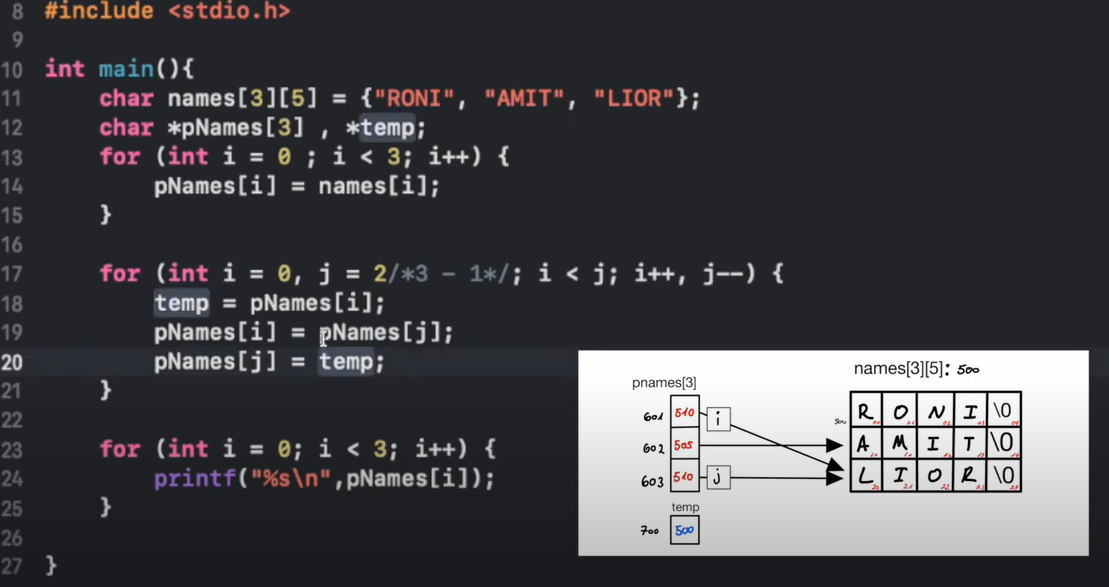
השם של המערך הוא גם מצביע אבל הוא קבוע למצבעים

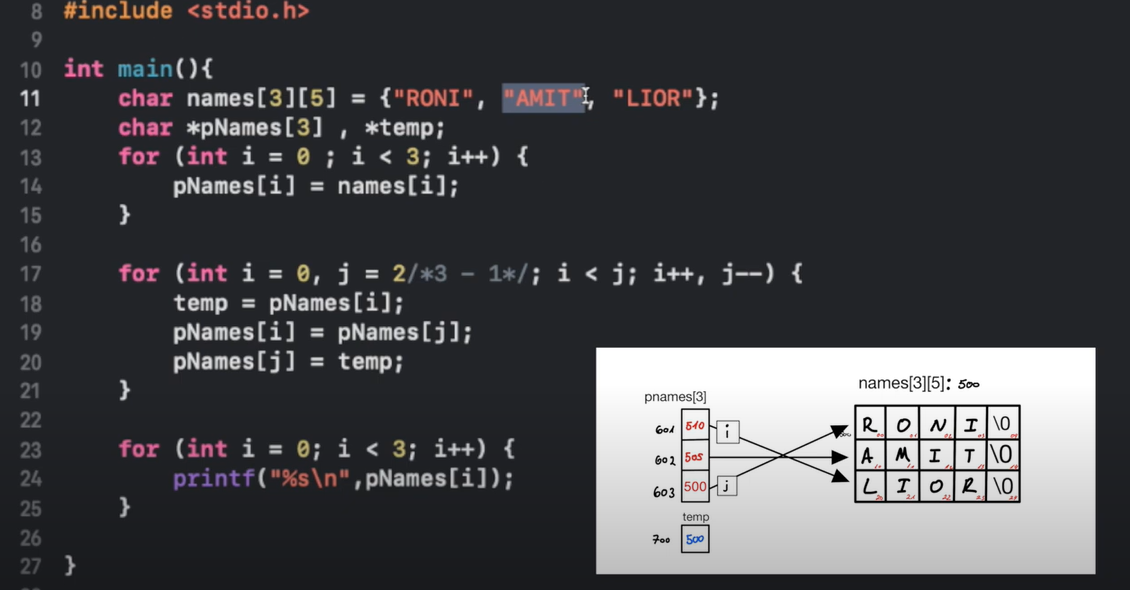
מתי משתמשים ?

* במערך דו ממדי לא שלם (שורות ועמודות לא אותו גודל )
* עבודה יותר נוחה במחרוזות מאשר אינדקסים









עד עכשיו כל פעם שיצרנו משתנה – יצרנו אותו על מחסנית נקרא זיכרון סטטי

המשמעות במחסנית היא שכל המשתנים מקומיים

כלומר מחסנית תומכת פקודת הכנסה והוצאה ממחסנית LIFO

הקצאת דינאמית סטטית מוקצית כבר בתחילת התוכנית וגודלה ידוע בזמן קומפילציה

מה הבעיה בזיכרון סטטי?

משתנה מוקצה בזמן קומפילציה – כשנגדיר מערך נהיה חייבים לתת לו גודל מסוים והוא יהיה חייב להיות ידוע בזמן קומפילציה ולכן לא נוכל ליצור מערך בגדול מסוים בהתאם לקלט של משתמש

הבעיה השניה – מערך קטן מדי

אין מספיק מקום, מערך גדול מדי זה אומר בזבזנו מקום בזיכרון

הפתרון לבעיות זה הקצאה דינאמית מוקצה על שטח HEAP על ערימה

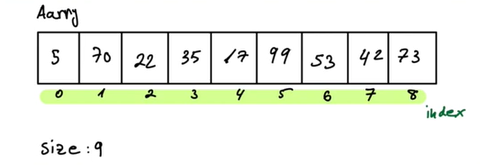
הוא אזור שמכיל זיכרון שהוקצה דינאמית זה אומר – שהאזור הזה משתנה בזמן ריצה בהתאם להקצאות ולשחרור הזיכרון שמתבצעים במהלך הריצה

אז בזמן ריצה אנחנו נקצה זיכרון אחרי שאנחנו יודעים מה הגדול שאנחנו צריכים

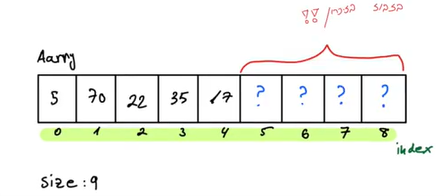
בהקצאת זיכרון דינאמי אנחנו יכולים להקצאות בדיוק את הגודל הדרוש לתוכנית לכן נהיה חייבים לשחרר את הזיכרון שהוא לא נחוץ יותר

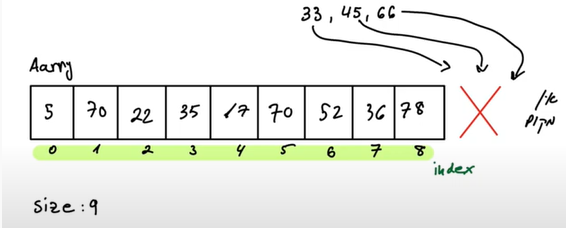
בשביל לעבוד על הקצאות דינאמית נצטרך לקרוא לספריה בשביל להשתמש במספר פונקציות

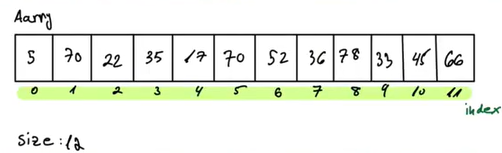
**פונקציה malloc**



בקשה לשנות את הגודל של המערך







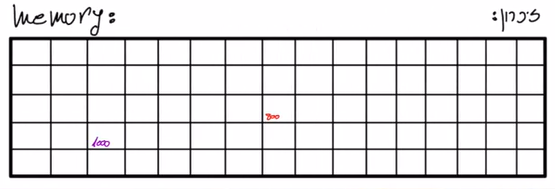
הגדלה של המערך – dynamic memory allocation

**malloc()= memory allocation**

הקצאה דינמית



משבצות ריקות = שטח פנוי בזיכרון



הכחול זה הכתובות התפוסות בזיכרון



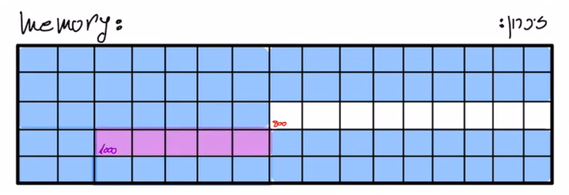
הפונקציה malloc מחזירה מצביע מסוג void אשר ניתן להמיר אותו למצביע מכל טיפוס

נצטרך להמיר לטיפוס שאנחנו צריכים

לכן הפונקציה מקבלת גם את גודל הזיכרון המבוקש בביטים ומחזירה כתובת לתחילת שטח הזיכרון שהוקצה

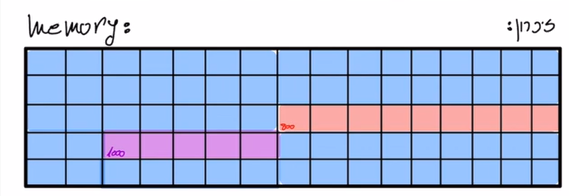
כלומר נעזר בפונקציה sizeof המחזירה את גודל הטיפוס בביטים





מתבצע המרה void ל int\* וכעת, ptr מחזיק את הכתובת שהוקצה בזיכרון במערכת





מה קורה אם פונקציה malloc לא מצליחה למצוא מקום בזיכרון לפי הגודל שבקישנו?

תחזיר לנו את הערך NULL - לכן כל פעם שנעשה הקצאה דינמית נהיה מחויבים לבדוק שאכן קיבלנו מקום בזיכרון

בנוסף, כאשר מקבלים הקצאת בזמן ריצה הוא מאתחל כל תא לערך זבל!!

**פונקציה free**

הזיכרון שמוקצה במחסנית הוא הוא מוקצה ע"י הקומפיילר ולכן הוא גם משוחרר אוטומטית בסיום התוכנית

בשונה מזיכרון דינמי – המוקצה ע"י המתכנת וזה אחריות של המתכנת לשחרר אם לא זה יהיה דליפת זיכרון

הזיכרון לא משחררת באופן אוטומטי!

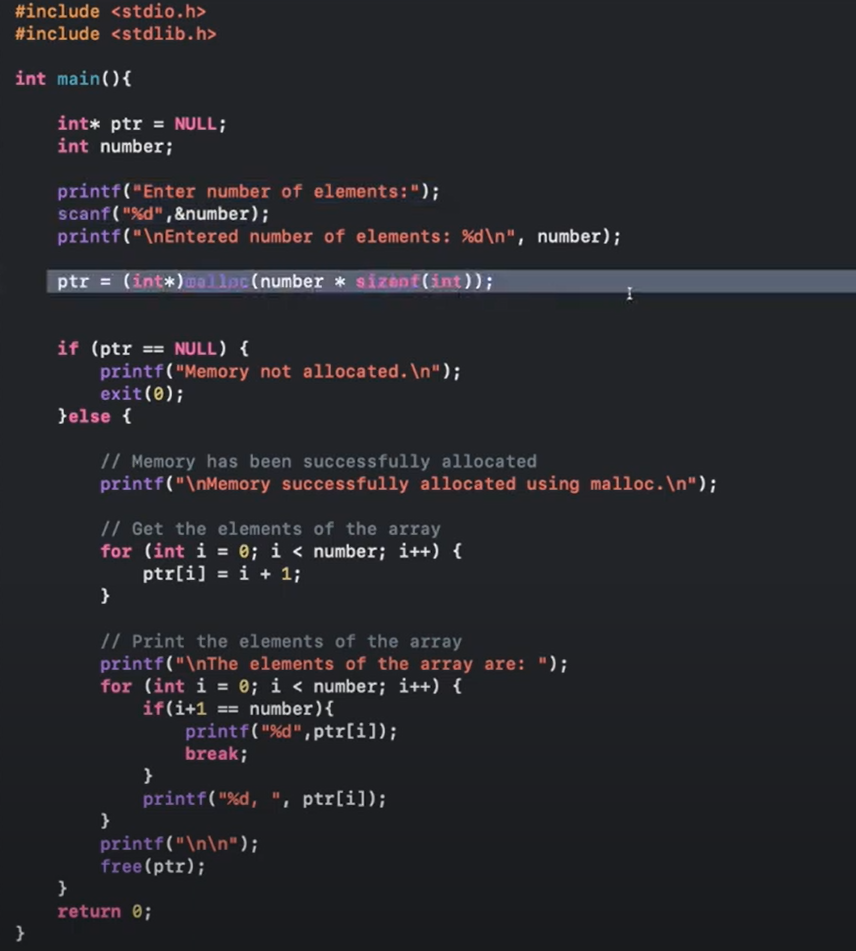
פונקציה free מקבלת את הכתובת של הבית הראשון שמוקצה ומשחררת את כל הזיכרון



כלומר- בתוך פונקציה נעביר את המצביע שמצביע על הכתובת של הבית הראשון בזיכרון

דוגמא איך מקצים זיכרון לפי קלט מהמשתמש מכניסים אלמנטים לזיכרון ומדפיסים אותם

**אנחנו מקצים ואנחנו משחררים**



ולכן כדאי -  שנוכל להשתמש בתוכנית ptr לצורך הקצאה מחדש

**פונקציה calloc()** = clear allocation - הקצאת זיכרון בזמן ריצה

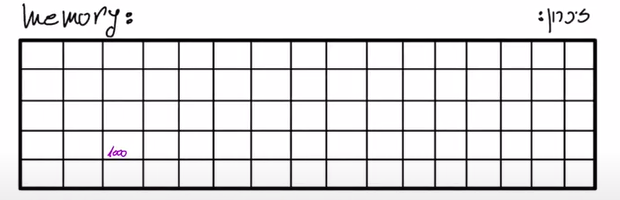


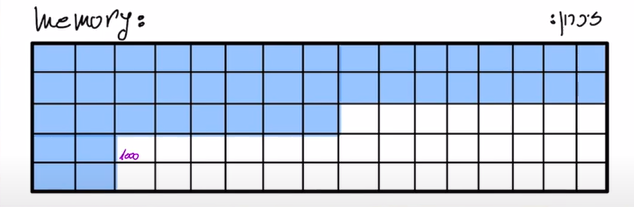
size מספר של אלמנטים כלומר גודל המערך

element\_size הוא קובע מה הגודל של כל תא – כלומר את גודל הבתים נעזר בפונקציית sizeof

כמו malloc אבל מקבלת שני ארגומנטים

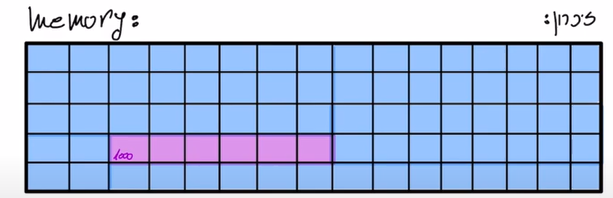
גם מחזירה טיפוס מסוג void





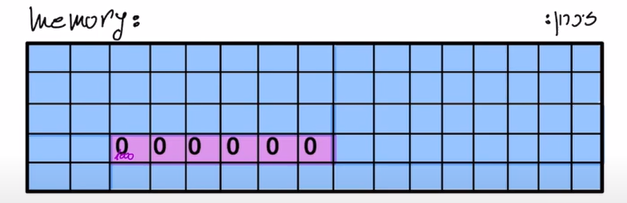


הגדרנו את ptr כ int \* ולכן נעשה המרה ל int\*

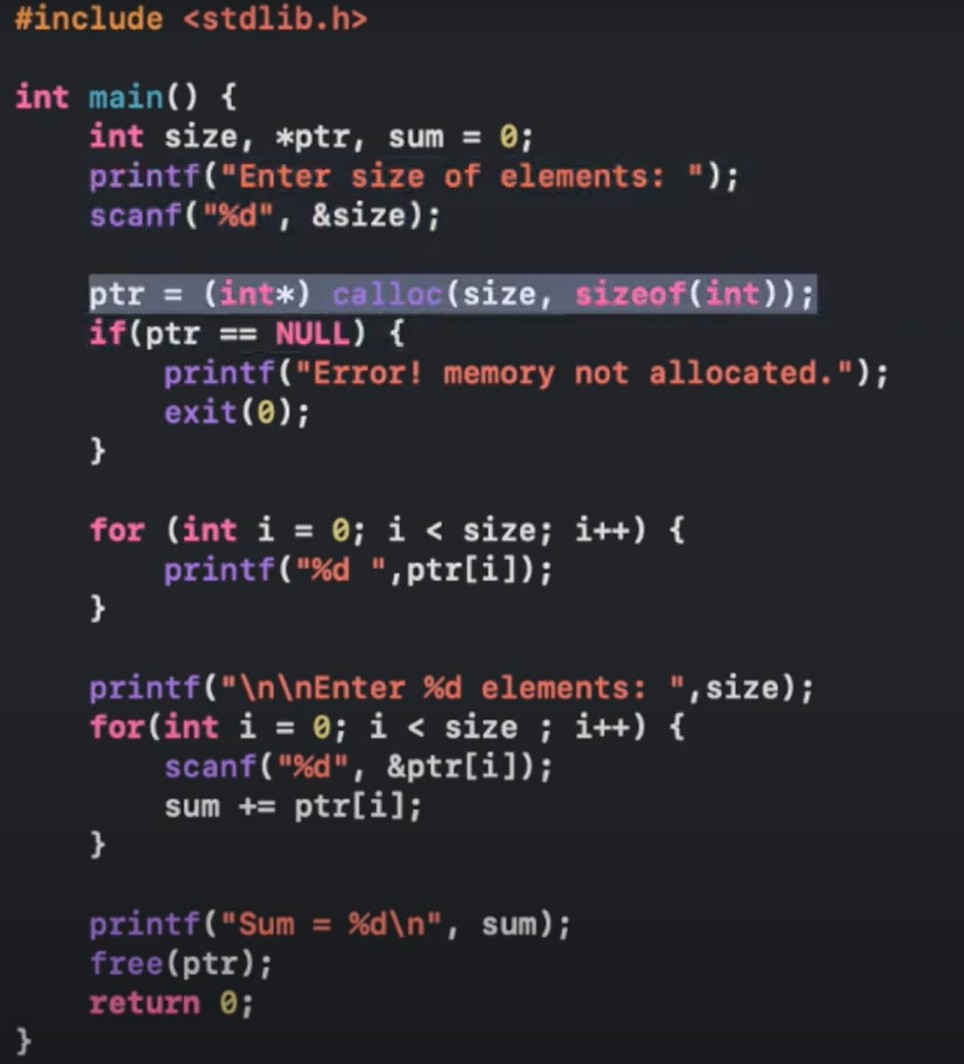


אותו דבר כמו malloc אם הערך הוא NULL זה אומר שלא הצליח להקצאות אחרת הוא מביא את הכתובת של הבית הראשון בזיכרון

**עוד משהו מאוד חשוב ב calloc הוא מאתחל כל מערך בזיכרון 0**



**בנוסף, זה פונקציה מאוד איטית ולא יעילה ולוקחת זמן ולכן לא משתמשים בה**



**פונקציה realloc() = re allocation**

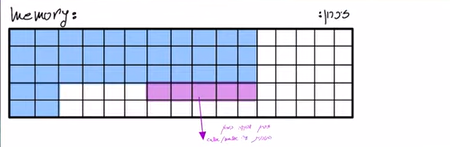
משמשת לשינוי דינמי של הקצאת זיכרון לזיכרון שהוקצה בתוכנית

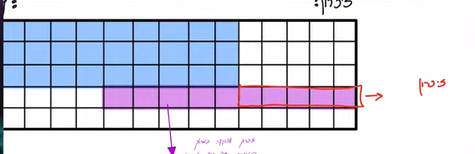
יתרונות של הקצאה דינאמית זה הגמישות לספק בדיוק את הגודל הדרוש

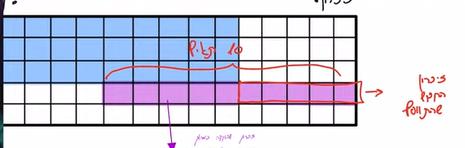
אם הקצנו באמצעות malloc or calloc והגודל שהשתמשנו לא מספיק עבורנו או שהשתמשנו רק בחלק מהזיכרון שזה בזבוז זיכרון נשתמש בפונקציה realloc בשביל להקצות באופן דינמי באופן החדש

הדגשים – גודל החדש = הוא גודל ישן + הגודל חדש

למשל אם היה לנו הקצאה בגודל 5 ואנחנו רוצים להגדיל אותו ל 10









ptr - זה המצביע שמצביע על הקצאה הישנה כלומר שהוקצה ע"י malloc or calloc

newSize – הוא הגודל החדש (הקטן או הגדול יותר)

אם הערך ptr הוא NULL היא תתנהג כמו malloc (אבל חובה! לקרוא ל malloc or calloc קודם!)

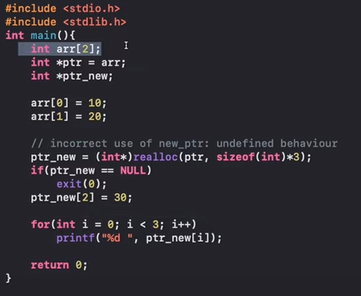
התהליך של realloc בודק אם יש מספיק מקום מרחיב את הקצאה הישנה אם לא מקצה מקום חדש בזיכרון לפי הגודל הישן + הגודל החדש

מעתיק את המציע הישן למקום החדש ומשחרר את הזיכרון של המקום הישן

הפונקציה מחזירה את הכתובת של הבית הראשון של הקצאה

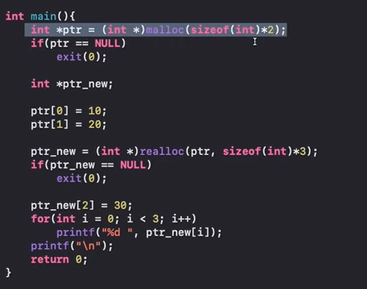
כאשר נקרא לפונקציה והקריאה תצליח – תחזיר את הבית הראשון אבל אם מחזירה NULL זה אומר שלא היה מספיק זיכרון להקצאה

דוגמא: מחזירה שגיאה בזמן ריצה



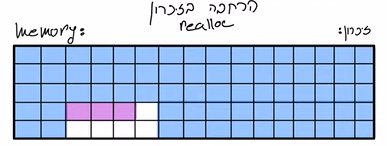
בגלל שלא קראנו לפונקציה malloc or calloc

תיקון



**מה חסר בתוכנית?**





אבל אם לא היה מקום

