**סוגי הדפסות**

* d% עבור טיפוסים מסוג int
* ld% עבור טיפוסים מסוג long int
* c% עבור טיפוסים מסוגchar ⬄ מקביל לפונקציהputchar()
* f% עבור טיפוסים מסוג float
* u% עבור טיפוסים מסוג unsigned
* lf% עבור טיפוסים מסוג double
* L% עבור טיפוסים מסוג long double
* s% עבור [מחרוזות](https://he.wikibooks.org/wiki/%D7%A9%D7%A4%D7%AA_C/%D7%9E%D7%97%D7%A8%D7%95%D7%96%D7%95%D7%AA)⬄מקביל לפונקציה PUTS
* p% עבור [מצביעים](https://he.wikibooks.org/wiki/%D7%A9%D7%A4%D7%AA_C/%D7%9E%D7%A6%D7%91%D7%99%D7%A2%D7%99%D7%9D)  - הדפסת כתובת
* %g הדפס הכי מקוצר שאפשר

נוכל לקבוע את תצוגת ההדפסה יותר בדיוק על ידי ציון של כמה ספרות נקצה שכל ההדפסה תהיה ואם מדובר במספר ממשי אז נוכל גם לבחור כמה ספרות אחרי הנקודה להציג.

נוכל לעשות זאת כך:

printf("%10.3f**\n**", 3.14159265);

printf("%10.3f**\n**", 3.1);

printf("%10.3s**\n**", "Hello");

במקרה כזה תוצאות ההדפסה יהיו:

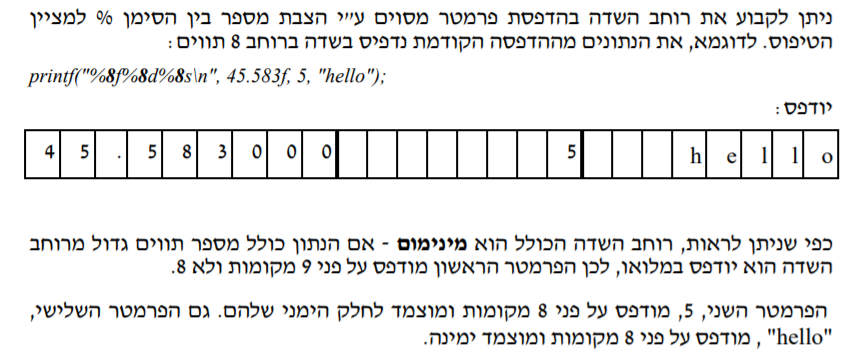
3.142

3.100

Hel

נוכל לראות שכיוון שרשמנו לו שיציג את התשובה על 10 מקומות ו-3 אחרי הנקודה אז הוא אכן עשה לנו זאת ועל מנת למלא את שאר המקומות כי חייבנו אותו להדפיס על 10 מקומות הוא מילא משמאל רווחים.

נוכל גם לראות שמבחינתו הנקודה העשרונית גם תופסת חלק מה-10 מקומות.

נוכל להראות כאן גם שבמחרוזת הספרות אחרי הנקודה אומרות כמה תווים לקחת מההתחלה של המחרוזת.

על מנת להצמיד את ההדפסה לצד שמאל עושים '-' מייד אחרי האחוז לדוגמא:  
 "%-8f" ידפיס על 8 תווים אך מוצמד לשמאל.

**סוגי קלט**

אם width הוא מספר, אז ייקלטו לכל היותר width תווים. לדוגמה:

scanf("%3d", &num)

אם נקליד יותר תווים הוא יקח את ה-3 הראשונים.

תקלוט לכל היותר 3 תווים. כך מאפיינים גם רוחב קליטה למחרוזות. לדוגמה:

char a[5];

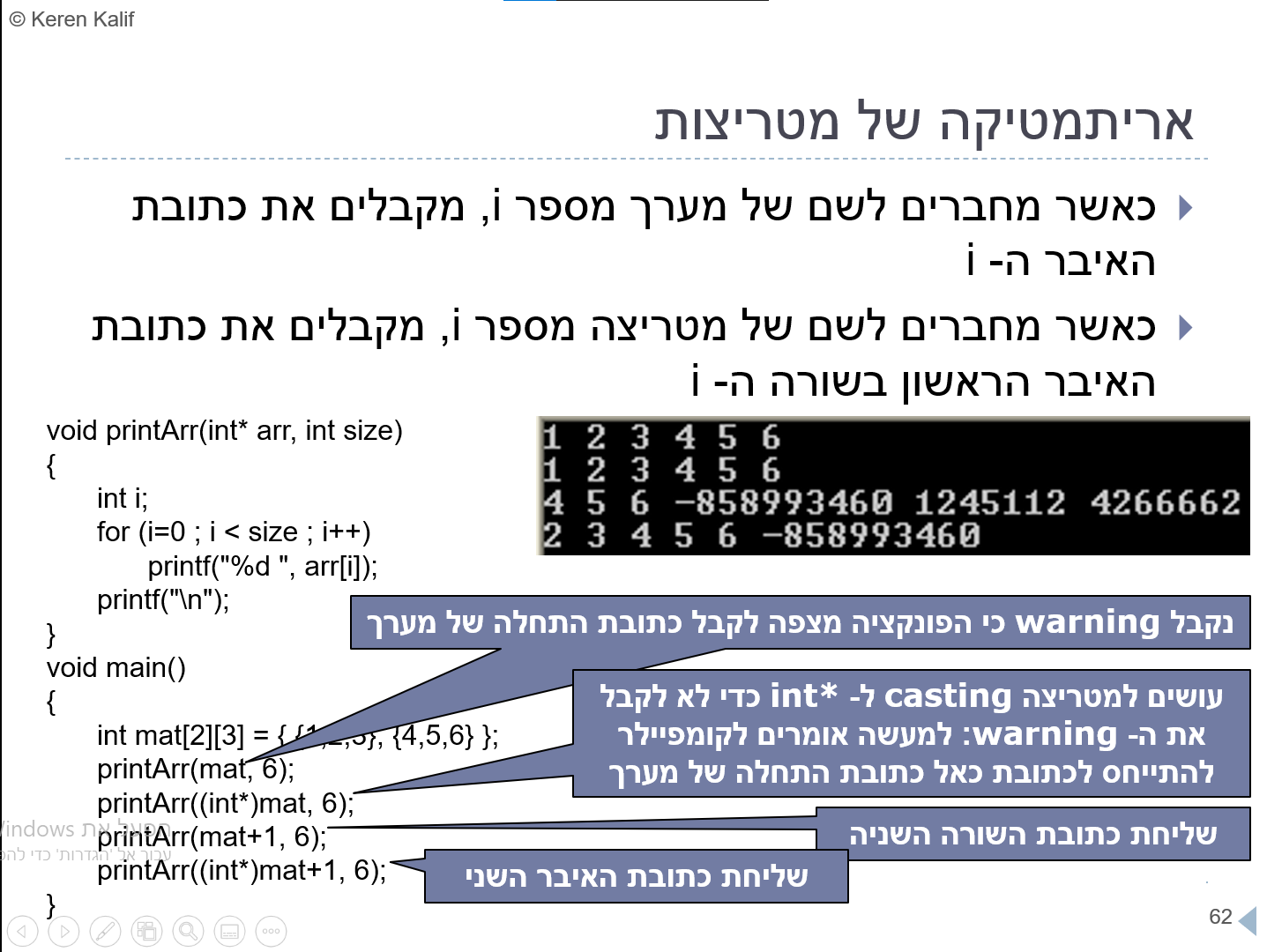
scanf("%4s", a);

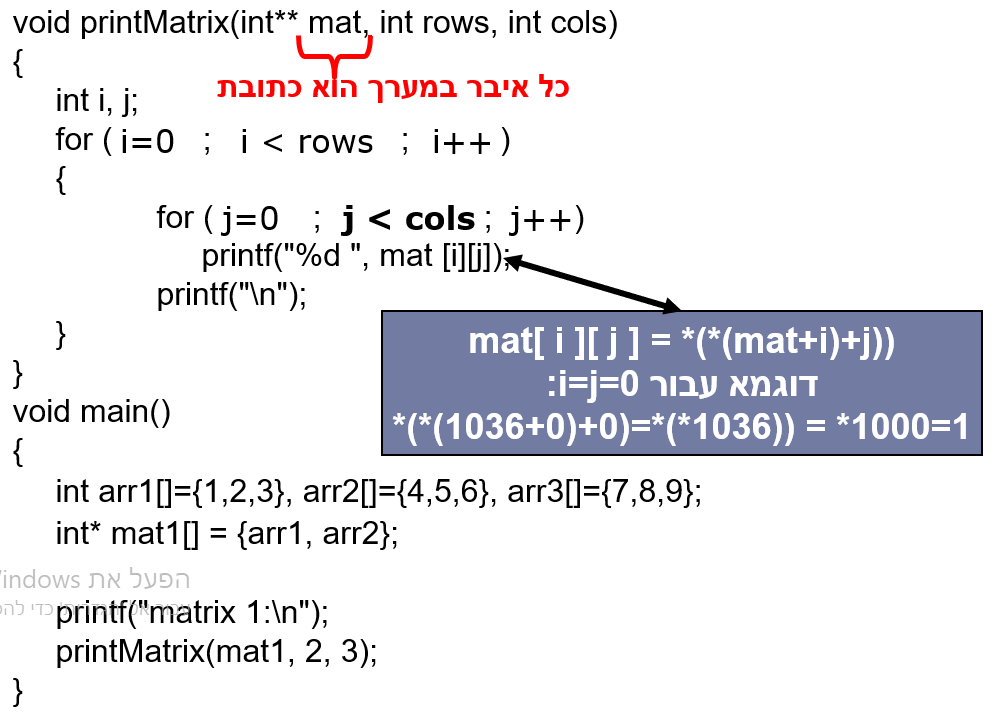
עשינו בכוונה 4 כי צריך להשאיר מקום ל-NULL . אם נקליד יותר הוא יקח את ה- 4 תווים הראשונים.

נזכור שעל מנת לקלוט מחרוזת נוכל להשתמש בפונקציה GETS.

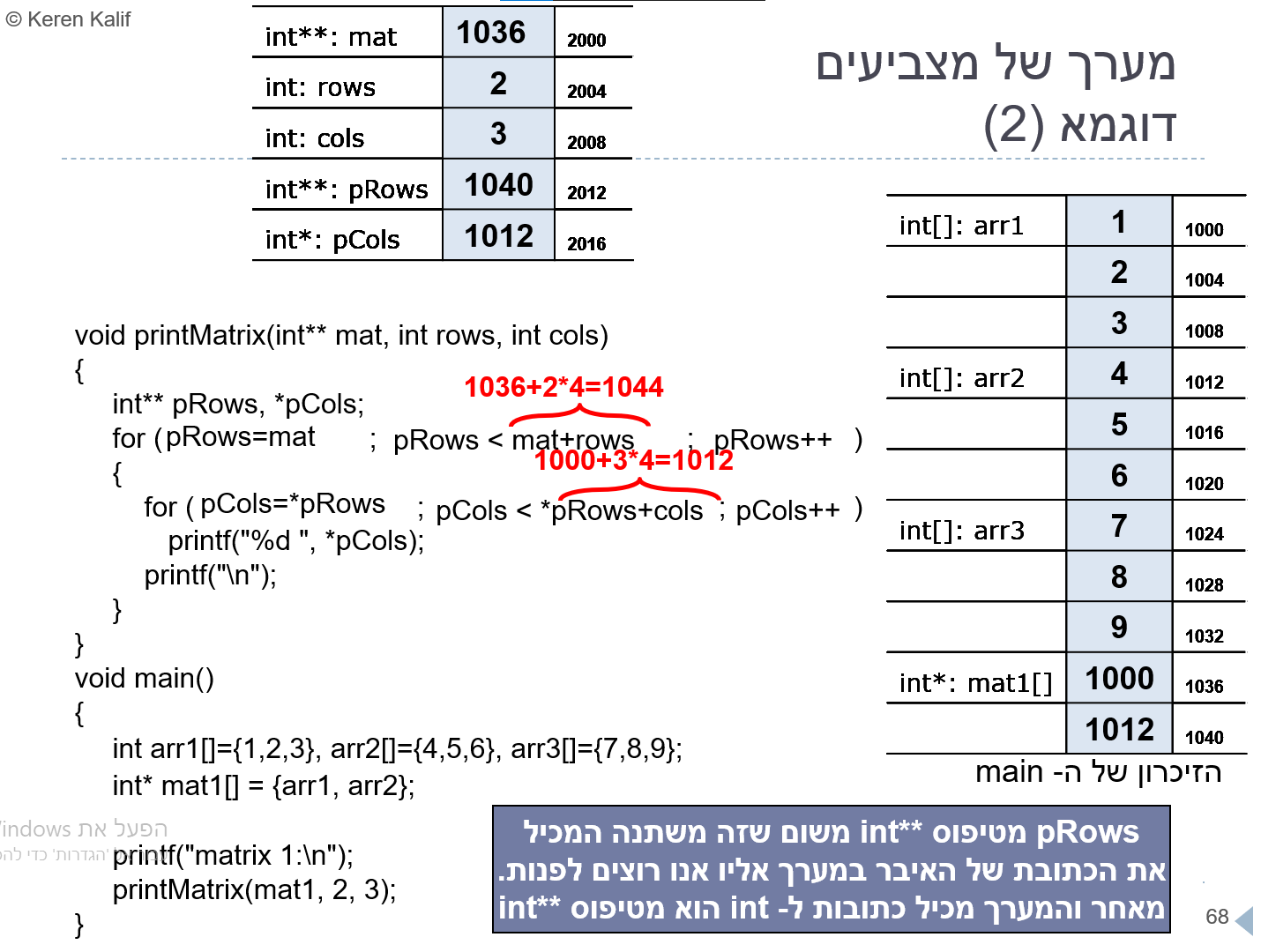
* **נוכל לראות האחוזים של כל טיפוס בדף של הקלטים**

**מטריצות מבחינת כתובות**

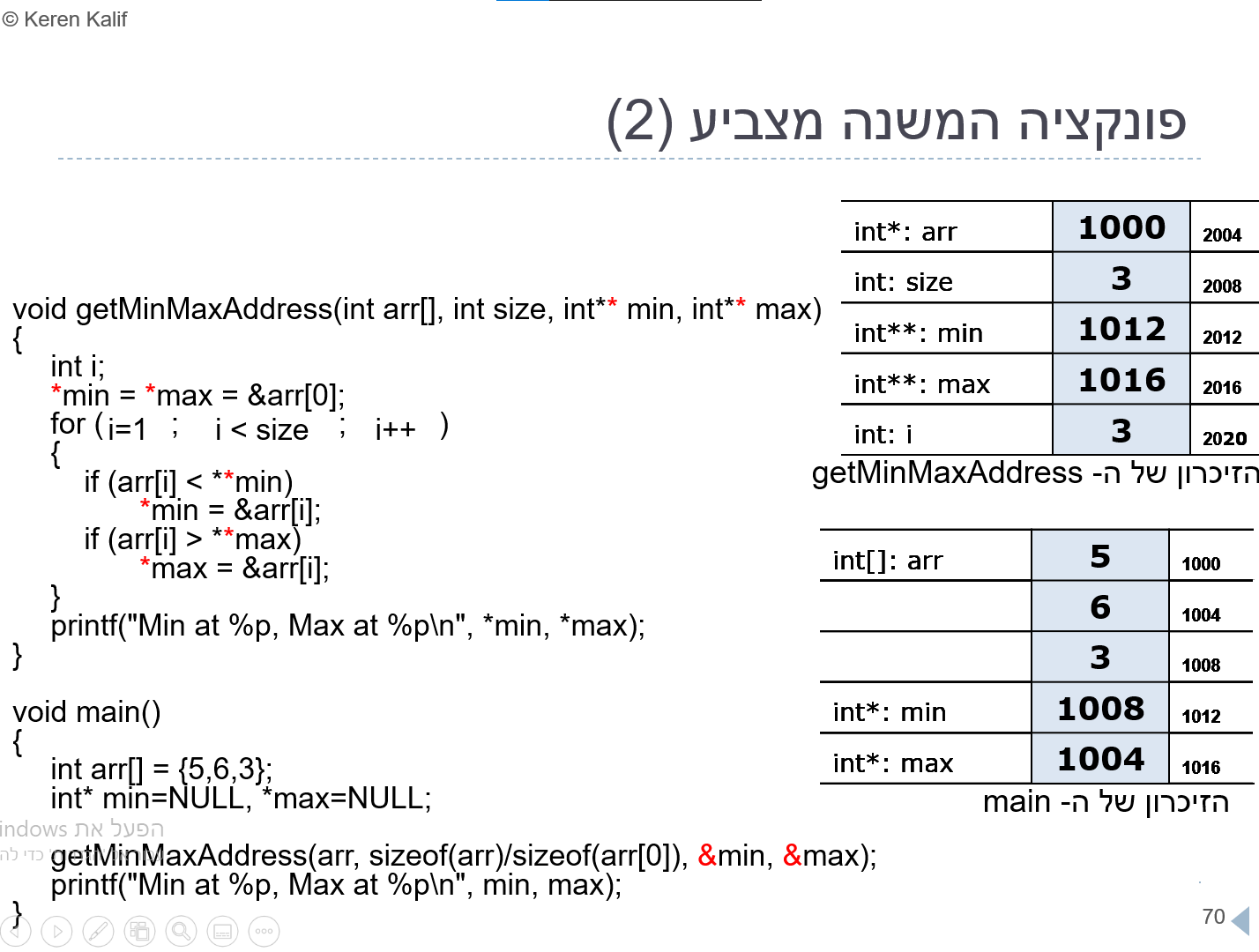


**הדפסת מטריצות בפונקציה:**

**הדפסה עם טיול פויינטרים:**



בפעם השנייה יהיה 1012 ולא 1000

**פונקציה המשנה פויינטר ב-MAIN:**

**sscanf + sprintf**

בשפת C קיימת אפשרות להתייחס למחרוזת כאל מקורות קלט ויעדי פלט ע"י שימוש בפונקציות sscanf או sprintf .

**sscanf**

משמשת לקריאת קלט ממחרוזת על פי פורמט נתון.

לדוגמא:

void main() {

char sentence[] = "my grade is: 100 and i happy";

char str[20];

int i;

sscanf(sentence, "%\*s%s%\*s%d", str, &i);

printf("%s -> %d",str, i);

}

החלק הראשון יהיה לאן נרצה לקלוט.

בחלק השני נייצג אילו מיקומים במחרוזת ניקח ומאיזה טיפוס הוא. למשל בדוגמא הזו נעשה %\*s משמעות הדבר הזה הוא לדלג על המילה הראשונה(כוכבית בין האחוז למייצג אומר דילוג כאן) לאחר מכן נעשה %s כיוון שנרצה לקלוט את המילה השניה שהיא grade לאחר מכן נבצע עוד דילוג ואז נקלוט איבר שהוא בעצם מספר ונייצג %d כדי להגיד לו שיסתכל על זה בתור int ולא בתור String.

בחלק השלישי אנחנו נגיד לו לאן לקלוט את מה שאמרנו. למשל בדוגמא הזו אנחנו נקלוט את הסטרינג(grade) אל תוך str ואת המספר את תוך i .

כך בעצם נוכל לקלוט מתוך string כל מיני משתנים.

בסוף יודפס: "grade -> 100".

**sprintf**

#include <stdio.h>

typedef char string[256];

int main()

{

string str = "yes",str2;

int num = 50;

sprintf(str2,"num = %d,str = %s",num,str);

puts(str2);

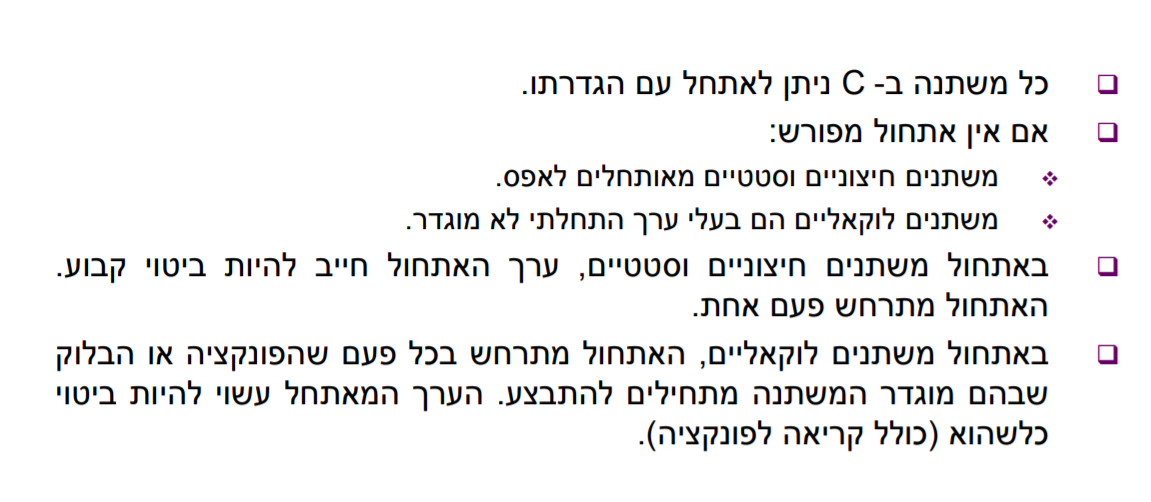
return 0;

}

בעצם sprintf הוא סוג של קליטה. למשל בקטע הקוד הבא אתה רושם לאיפה את רוצה לקלוט את הסטרינג הבא(str) בחלק השני אתה רושם את הסטרינג אבל אתה יכול לשים אחוזים של int/double/string וכו'... ובחלק השלישי את מה שתרצה להכניס לאחוזים.

בסוף str2 יהיה שווה: "num = 50, str = yes".

**משתנה STATIC**

משתנה סטטי לוקאלי: משתנה לוקאלי לכל דבר, אלא שהוא שומר על ערכו בין קריאות שונות לאותה הפונקציה.



\*\* אם יש מערך סטטי לא מאותחל למשל static int arr[20]; אז כל איברי המערך יהיו 0.

**\*\*\* אתחול של מערך רגיל באפס יתבצע כך: {0} אך נוכל לעשות זאת רק בהגדרתו!!!**

**פעולות על מחרוזות**

* **Strcmp – השוואה בין מחרוזות**

**Int strcmp(const char\* s1, const char\* s2)**

הפונקציה עושה חיסור בין הערכים האסקים של כל סטרינג כלומר s1-s2 = res את והפונקציה מחזירה את res.

למשל:

#include <stdio.h>

#include <string.h>

void main()

{

char s[20] = "helo", t[20] = "hell";

int i = strcmp(s, t);

printf("%d", i);

}

הפלט יהיה: 3

מדוע? כיוון ש-o יותר גדול מ- l ב-3 ולכן החיסור שלהם יתן לנו 3.

* **Stricmp – השוואה בין מחרוזות ללא התייחסות לאותיות גדולות או קטנות**

**Int stricmp(const char\* s1, const char\* s2)**

הפונקציה עושה חיסור בין הערכים האסקים של כל סטרינג ללא התייחסות לאותיות גדולות או קטנות.

* **Strcpy העתקת מחרוזת אחת למחרוזת שנייה**

**Char\* strcpy(char\* s1, const char\* s2)**

הפונקציה מקבלת מחרוזת קבועה ומעתיקה אותה למחרוזת שבאיבר הראשון. כלומר s2 יועתק אל s1.

אם היה משהו ב-s1 הוא ידרס ויכיל את s2.

* **Strlen – מחזירה את אורך המחרוזת**

**Size\_t strlen(const char\* str)**

הפונקציה מקבלת מחרוזת ומחזירה את אורכה.

* **Strcat – שרשור מחרוזת אחת למחרוזת שנייה**

**Char\* strcat(char\* dest, const char\* src)**

נשים לב שהאיבר שמשרשרים למחרוזת חייב להיות קבוע.

**דוגמא לשימוש בפונקציות strcpy ו- strcat :**

#include <string.h>

#include <stdio.h>

typedef char string[256];

void main(){

string str;

strcpy(str,"Hello " );

strcat( str, "from " );

strcat( str, "noam" );

puts(str);

}

הפלט יהיה: "hello from noam"

**לפונקציות  strcmp, strcpy ו- strcat יש פונקציות מקבילות הפועלות על מספר נתון של תווים:**

**strncmp משווה בין 2 מחרוזות מספר נתון של תווים לכל היותר, strncpy מעתיקה מספר נתון של תווים לכל היותר ממחרוזת אחת לשנייה, ו- strncat משרשרת מספר נתון של תווים לכל היותר ממחרוזת אחת לשנייה.**

int strncmp(const char \*str1, const char \*str2, size\_t n)

char\* strncpy(char\* dest, const char\* src, size\_t n)

char \*strncat(char \*dest, const char \*src, size\_t n)

בפונקציה strncmp ה-n הוא מקסימאלי עד איפה לקלוט אבל אם יהיה NULL באמצע הסטרינג עוד לפני התו השמיני הוא יעצור ב-NULL.

בפונקציה strncat בחלק השלישי נקבל כמה תווים מהמחרוזת src נשרשר ל-dest אם נגיד לו לשרשר יותר תווים מאורך המחרוזת של src הוא יקח עד הנאל. בנוסף נשים לב שאנחנו מחוייבים לדאוג שיהיה מספיק מקום במערך של היעד על מנת להכיל את השרשור.

**פונקציות לפעולות חיפוש טקסט**

* **Strchr מציאת מיקום של המופע הראשון של תו מסוים במחרוזת**

**Char\* strchr(const char\* s,int c**) //באותה מידה היה אפשר לרשום צ'אר

מחזירה פויינטר למיקום הראשון שבו הוא פוגש את הצ'אר

* **strrchrמציאת מיקום של תו מסוים במחרוזת החל מהסוף :(reversed)**

**Char\* strrchr(const char\* s,int c**) //באותה מידה היה אפשר לרשום צ'אר

* **strstrמציאת מיקום של תת-מחרוזת במחרוזת:**

**Char\* strrchr(const char\* s, const char\* sub\_str**)

אם יש את התת מחרוזת המחרוזת אז הפונקציה תחזיר מצביע למיקום הראשון במחרוזת בו מתחילה התת מחרוזת. אם אין את המחרוזת שמחפשים הוא יחזיר NULL.

* **strtok מציאת מילה במחרוזת, עפ"י תווי פיסוק נתונים. תווי הפיסוק מוחלפים בתו מסיים מחרוזת, והמילה מוחזרת.**

**char \*strtok(char \*str, const char \*delim)**

הפרמטר הראשון הוא מחרוזת והפרמטר השני הוא אוסף תווים (מחרוזת) המהווים סימני פיסוק במחרוזת הראשונה. פונקציה זו פועלת בשני אופנים:

* בפעם הראשונה שהיא נקראת מעבירים לה את שני הפרמטרים. היא מחפשת את תו הפיסוק הראשון במחרוזת, מחליפה אותו בסימן סוף מחרוזת ('\0') ומחזירה מצביע לתת-המחרוזת שנוצרה.
* בפעמים הבאות ש- strtok נקראת, הפרמטר הראשון הוא NULL  והיא ממשיכה לחפש סימני פיסוק מהמקום האחרון בו הפסיקה. היא מחזירה שוב מצביע לתת-המחרוזת החדשה שנוצרה (אם הייתה כזו).

#include<string.h>

Typedef char string[256];

Void main(){

String line = "first,second;third?fourth";

Char\* ptr = strtok(line,",;?");

While(ptr != NULL){

Printf(("%s\n",ptr);

Ptr = strtok(NULL,",;?");

}

}

**הפונקציה תדפיס:**

first

second

third

fourth

**הערות:**

1. יש לשים לב ש- strtok() משנה את המחרוזת המקורית המועברת לה ע"י החלפת סימן הפיסוק המתאים בתו מסיים מחרוזת.

2. לצורך שמירת המקום האחרון בו הפסיקה, strtok() מגדירה משתנה מקומי סטטי ש"זוכר" את כתובת התו האחרון שנסרק במחרוזת.

3. במידה ויש מספר תווי פיסוק עוקבים, strtok() מזהה זאת ומדלגת עליהם, עד למציאת תת-מחרוזת המכילה לפחות תו אחד שאינו תו פיסוק.

* **strspn פועלת כמו  strtok, אך לא מכניסה סימן סוף מחרוזת. במקום זאת, היא מחזירה את אינדקס התו הבא שלאחר סימני הפיסוק.**

**size\_t strspn(const char \*str1, const char \*str2)**

**הפונקציה מחזירה את האינדקס לאחר התת מחרוזת שהבאנו(str2). אבל חייב להתחיל מההתחלה כלומר אם למשל יש את הקוד הבא:**

int main () {

int len;

const char str1[] = "ABCDEFG0198714";

const char str2[] = "CD";

len = strspn(str1, str2);

printf("Length of initial segment matching %d\n", len );

return(0);

}

**קוד זה ידפיס לי:**

**Length of initial segment matching 0**

**כיוון שלא התחלנו מההתחלה אם היינו רוצים את האינדקס שאחרי CD היינו צריכים גם לרשום את מה שלפני כלומר "ABCD" ואז זה הי מדפיס לנו 4. כלומר אנחנו רואים שהוא חייב להתחיל מההתחלה.**

* **strpbrk מקבלת כפרמטרים מחרוזת ותת-מחרוזת ומחזירה מצביע למופע הראשון של תו כלשהו מתת-המחרוזת במחרוזת, ואם לא נמצא מופע כזה מוחזר NULL.**

**char\* strpbrk(const char \*str1, const char \*str2)**

**memcmp + memset**

**memcmp**

**int memcmp(const void \*str1, const void \*str2, size\_t n)**

פונקציה זו מקבלת מצביעים למקומות בזכרון ומקבלת מספר המייצג כמה בתים להשוות.

הפונקציה עוברת על n הבתים מהמצביע הראשון וסוכמת את הערכים האסקים שיש שם ועושה אותו דבר עבור n הבתים הראשונים מהמצביע השני לאחר מכן היא עושה פעולת חיסור:

sum(str1) - sum(str2) ומחזירה את התוצאה.

\*\*\* אם אנחנו רוצים להשוות Int נצטרך לעשות כפול sizeof(int) במספר הבתים שנרצה להשוות וכו'..

**memset**

**void \*memset(void \*str, int c, size\_t n)**

פונקציה זו מקבלת מצביע למקטע זכרון, char לשים בכל בית, כמה בתים לעבור מהמצביע.

בארגומנט השני נוכל להעביר לו או ערך האסקי או פשוט את הצ'אר עצמו.

#include <stdio.h>

#include <string.h>

int main () {

char str[50];

strcpy(str,"This is string.h library function");

puts(str);

memset(str,'$',7);

puts(str);

return(0);

}

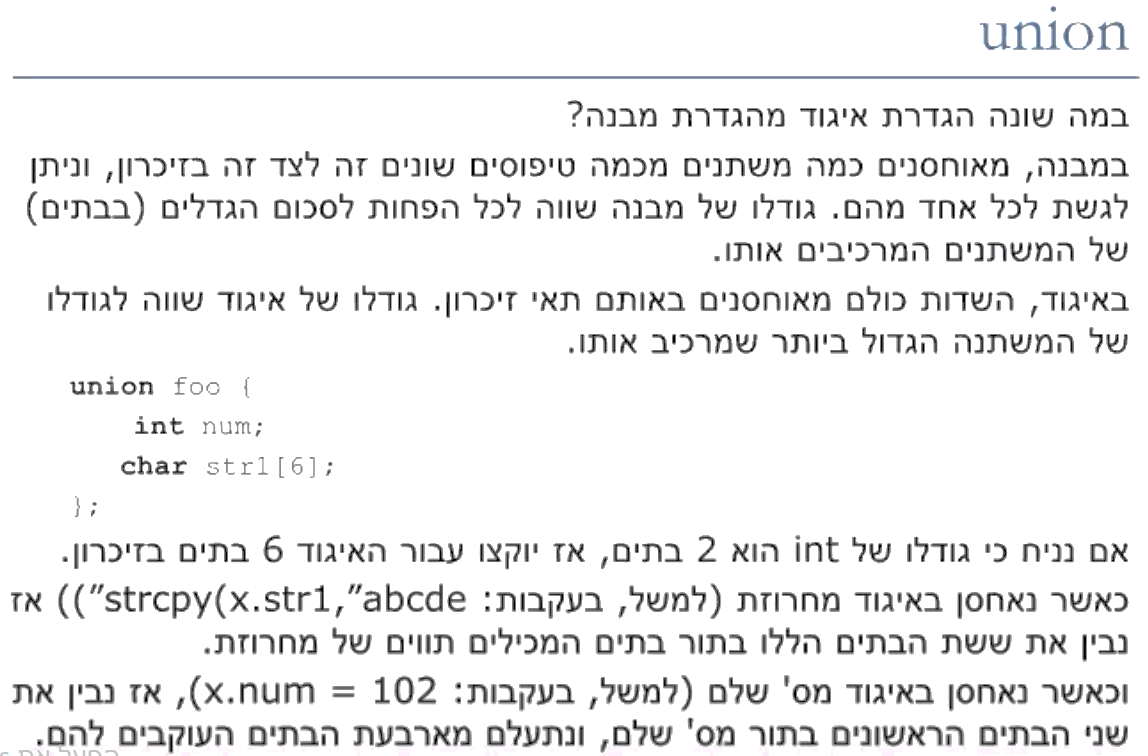
**הפלט יהיה:**

This is string.h library function

$$$$$$$ string.h library function

כיוון שלקחנו 7 תווים מתחילת המחרוזת ושמנו בכל בית את התו '$'.

**Union + structs**

Union נכתב כמו struct אך אינו מתנהג כמוהו מבחינת הזכרון.

נראה דוגמא למה הכוונה:

union Data }

int i;

float f;

char str[20];

};

void main(){

union Data data;

data.i = 10;

data.f = 220.5;

strcpy( data.str, "C Programming");

printf( "data.i : %d\n", data.i);

printf( "data.f : %f\n", data.f);

printf( "data.str : %s\n", data.str);

}

הפלט יהיה:

data.i : 1917853763 //כלומר לא צפוי

data.f : 4122360580327794860452759994368.000000// כלומר לא צפוי

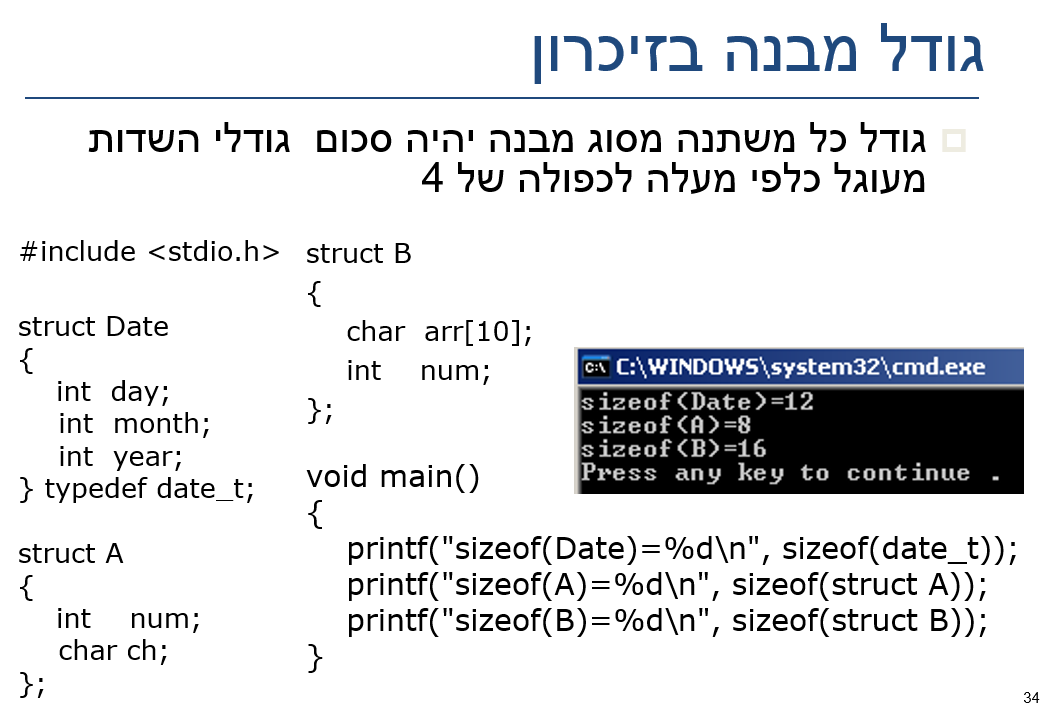
data.str : C Programming //רק הדבר האחרון ששמנו בבתים האלו יהיה

לפי מה שראינו הדבר האחרון שאנחנו שמים דורס את מה שהיה לפני ושם את עצמו. אם ננסה להדפיס את מה שיש במשתנה אחר של האיגוד נקבל משהו לא צפוי למשל כמו בדוגמא, הכנסנו אחרון סטרינג וכשניסינו להדפיס את data.i/data.f קיבלנו משהו לא צפוי כי מה שיש בבתים האלו כרגע זו מחרוזת. לכן חשוב מאוד לזכור איזה טיפוס הכנסנו אחרון ועיצד להתייחס לבתים הללו.

בנוסף לכך ישנו עוד הבדל בין struct ל-union והוא בעת ביצוע sizeof() .

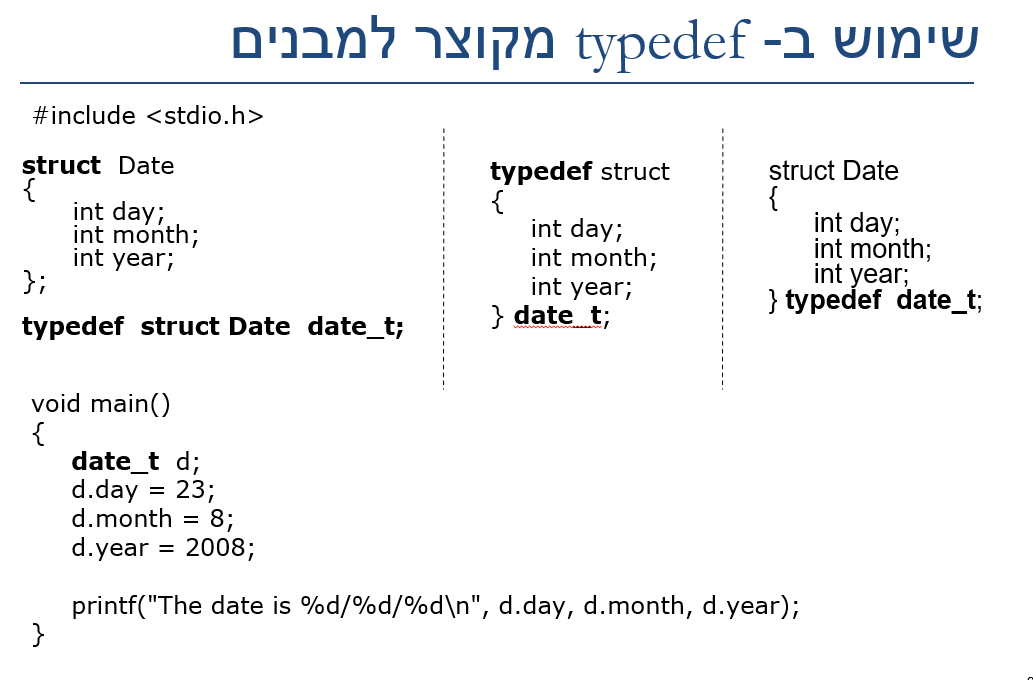
אם נבצע sizeof(struct…) נקבל את סכום גודלי השדות מעוגל כלפי מעלה לכפולה של 4.

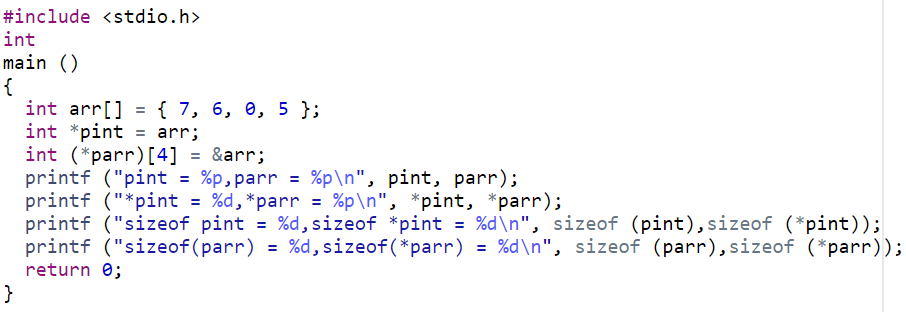
לעומת זאת אם נעשה sizeof(union…) נקבל את ה-sizeof של המשתנה הכי גדול בתוך האיגוד למשל בדוגמא אם היינו עושים sizeof(data) היינו מקבלים 20.



אם כבר הזכרנו מבנים... נראה רק כמה צורות לכתיבת מבנה והגדרתו ונראה גם כמה סגנונות לtypedef- למבנה.





**שאלות על מצביעים**

מה יהיה הפלט?

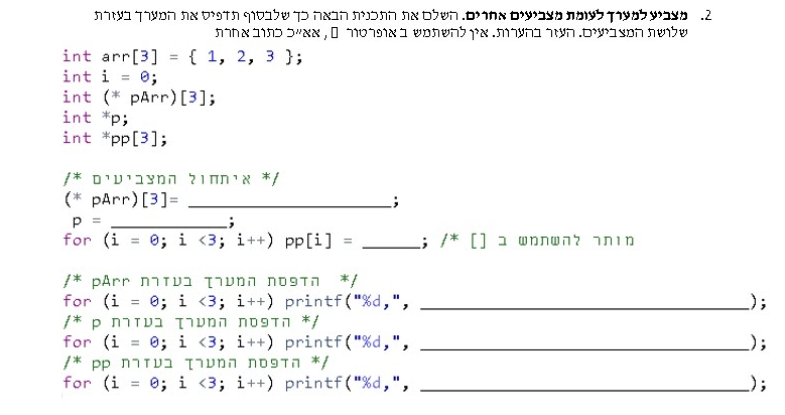
**pint = 1000, parr = 1000**

**\*pint = 7, \*parr = 1000**

**sizeof pint = 8, sizeof \*pint = 4**

**sizeof parr = 8, sizeof \*parr = 16**

בתכנית זו אנחנו יכולים לראות טוב את הסינטקס int (\*parr)[4] והוא מקבל &arr אנחנו יכולים לראות כאן את ההמחשה של הסינטקס שהוא בעצם מצביע לכל בלוק הנתונים ואם נעשה 1+ הוא יקפוץ לכתובת 1016. גם אנחנו יכולים לראות זאת בסוף שעושים sizeof(\*parr) אנחנו רואים שנקבל 16 כגודל כל המערך.



int main ()

{

int arr[3] = {1,2,3};

int i = 0;

int (\*parr)[3];//הסינטקס הזה אומר שזה מצביע שיכול להצביע רק על מערכים בגודל 3

int\* p;

int\* pp[3];

(\*parr)[3] = &arr; //כיוון שזה & אז זה יצביע על כל בלוק הנתונים. אם היינו עושים +1 היינו מקבלים את כתובת 1012 כלומר הכתובת שלאחר המערך

p = arr;

for ( i=0;i< 3 ;i++) {

pp[i] = &arr[i];

}

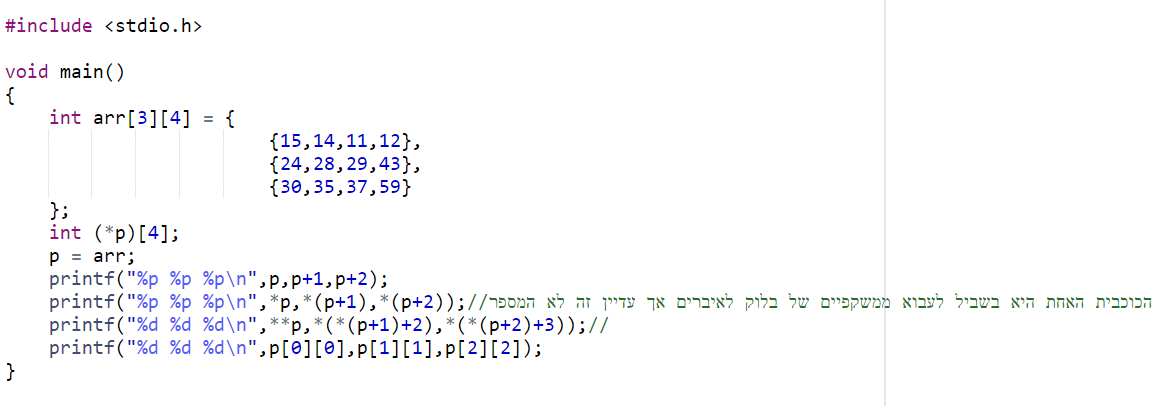
for(i=0;i<3;i++)printf("%d", \*(\*parr+i) );

for(i=0;i<3;i++)printf("%d", \*(p+i));

for(i=0;i<3;i++)printf("%d", \*\*(pp+i));

return 0;

}



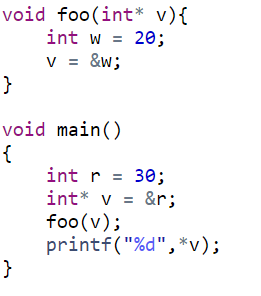
הפלט יהיה:

1000 1016 1032

1000 1016 1032

15 29 59

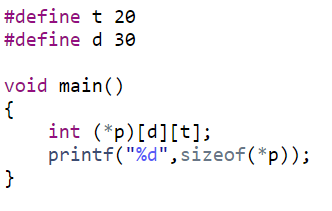
15 28 29



**מה התכנית תדפיס?**

1. 20
2. 30
3. שגיאת הידור
4. שגיאת קומפילציה

התכנית תדפיס 30 כי V של הפונקציה הוא רק העתק! אם היינו משנים את התוכן של המצביע אז זה באמת היה היה משנה את r ל-20.

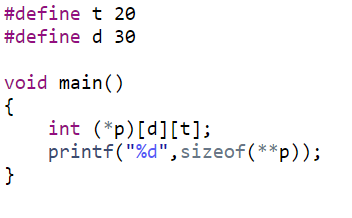
**מה יהיה פלט התוכנית?**

**הפלט יהיה**: 2400

מדוע? כיוון שזהו מצביע למטריצה!

זוהי לא המטריצה עצמה!

אז כעושים כוכבית אחת בעצם מגיעים לכל המטריצה!

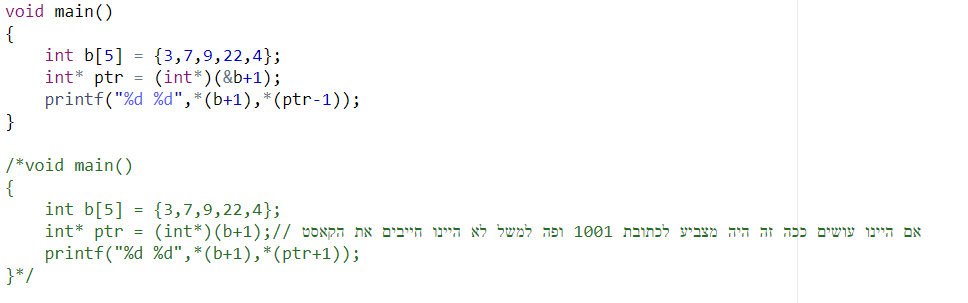
****

**מה יהיה פלט התוכנית?**

**הפלט יהיה**: 80

מדוע? כיוון שכוכבית אחת היא בעצם נכנסת לתוכן של המצביע כמו שראינו בתרגיל הקודם ואז זה מצביע לכל המטריצה. כוכבית נוספת תכנס להצביע על שורה ולכן sizeof של שורה הוא:

20\*4 = 80



**מה יהיה הפלט?**

**הפלט יהיה**: 7 4

מדוע? נסביר את התוכנית.  
  
בשורה השנייה אנחנו רואים את ה-& מה הוא אומר? הרי כבר b עצמו כתובת. ה-& אומר הכתובת של כל  
הבלוק של b כלומר מהכתובת 1000 עד הכתובת 1016. לאחר שזה &b אנחנו עושים +1 אז הוא יקפוץ  
הסייז אוף שלו וכיוון שה-SIZEOF שלו הוא 20 אז מהכתובת 1000 הוא יקפוץ לכתובת 1020 ויצביע מעכשיו על כל הבלוק מ-1020 עד 1036. אבל אז אנחנו עושים לו קאסטינג ל-int\* ולכן מעכשיו הוא יצביע על איבר בודד אך עדיין על הכתובת 1020.

בשורה השלישית אנחנו רואים \*(b+1) זה ידפיס לנו 7 כיוון ש-b זה כתובת 1000 ו+1 זה 1004 ואז התוכן יתן לנו 7. לאחר מכן אנחנו רואים \*(ptr-1) זה ידפיס לנו 4 כיוון ש-ptr כרגע מצביע על כתובת 1020 ואז 1- יתן לנו את כתובת 1016(בגלל הקאסטינג הוא ידע לרדת רק אחד פה ולא ירד את כל הבלוק) ואז עושים תוכן של 1016 וזה נותן לנו את 4.