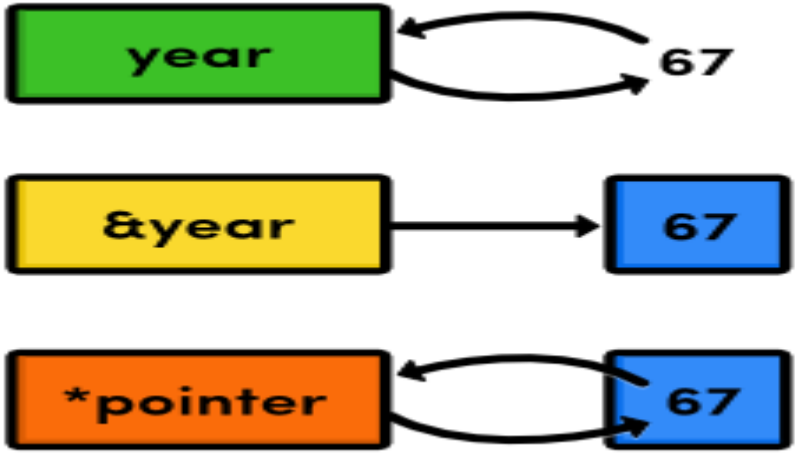
קורס יסודות התכנות בשפת C

פרק 9

מצביעים וכתובות בזיכרון Pointers



A variable transparently stores a value with no notion of memory addresses.

The reference operator returns the memory address of a variable.

The dereference operator accesses the value stored in a memory address.

ד"ר שייקה בילו
יועץ ומרצה בכיר למדעי המחשב וטכנולוגית מידע
מומחה למערכות מידע חינוכיות, אקדמיות ומנהליות

חזרה – קלט/פלט של מחרוזות

- ניתן לקלוט ולהדפיס את תווי המחרוזת בזה-אחר-זה, בעזרת הפונקציות:

scanf( ) , printf( )

- ניתן גם לבצע קליטה והדפסה של מחרוזת שלמה באמצעות הפונקציות:

getchar( ), putchar( )

חזרה - אפשרות לקלט/פלט של מחרוזות

- פקודת קלט נוספת למחרוזות שקולטת מחרוזת תווים כולל רווח:

```
gets(str);
```

- הפונקציה פועלת בדיוק כמו scanf(“%s”, str) , כלומר קולטת מחרוזת שלמה מהמסך.
- היתרון – המהירות, הכתיבה הקצרה יותר והנוחות הנגזרת מכך.
- החיסרון - לא ניתן להגביל את אורך מחרוזת הקלט, במידה והמשתמש יכניס מחרוזת ארוכה מידי התוכנית "תעוף".
- פקודת פלט תואמת המבצעת הדפסה וירידת שורה:

```
puts(str);
```

- הפונקציה פועלת בדיוק כמו printf(“%s\n”, str) , כלומר מדפיסה מחרוזת שלמה ולאחריה מבצעת ירידת.

חזרה - הספריה string.h – פונקציות לדוגמא

- פונקציה המבצעת מציאת אורך מחרוזת (לפי מקום התו '\0'):

```
int strlen(char str[ ]);
```

האורך לא כולל את התו '\0'

```
len = strlen(my_string);
```

השימוש מתבצע כך:

פונקציה המבצעת השוואה בין מחרוזות:

```
int strcmp(char str1[ ], char str2[ ]);
```

מחזירה 0 אם המחרוזות זהות, כלומר שוות בכל תו עד לתו '\0'

אחרת מוחזר מס' שונה מ-0: חיובי (1) אם המחרוזת הראשונה גדולה יותר

לקסיקוגרפית - כלומר לפי סדר מילוני, סדר המופיע בטבלת ASSCII

ושלילי (-1) אם היא קטנה יותר.

חזרה - הספריה string.h – פונקציות לדוגמא

- פונקציה המבצעת הפיכה של מחרוזת מהסוף להתחלה:

```
void strrev(char str[ ]);
```

- פונקציה המבצעת העתקת מחרוזת למחרוזת אחרת:

```
void strcpy(char target [ ], char source[ ]);
```

- פונקציה המבצעת שרשור מחרוזות:

```
void strcat(char str1[ ], char str2[ ]);
```

חזרה - הספריה string.h – פונקציות לדוגמא

- פונקציה המבצעת חיפוש ההופעה הראשונה של תו במחרוזת ומחזירה את שאר המחרוזת מהמופע הראשון של התו:

```
char strchr(char str[ ], char ch);
```

- פונקציה המבצעת חיפוש ההופעה הראשונה של מחרוזת במחרוזת אחרת ומחזירה את שאר המחרוזת מהמופע הראשון של המחרוזת:

```
char strstr(char str[ ], char find[ ]);
```

- פונקציה המבצעת חיפוש של ההופעה הראשונה של מחרוזת מסוף המחרוזת ומקצצת מהמקום בו נמצא המופע הראשון

```
char strtok(char str[ ], char find[ ]);
```

חזרה - הספריה string.h – פונקציות לדוגמא

- פונקציית השוואה נוספת מחזירה תשובות כמו strcmp:

```
int strcoll (char str1[ ], char str2[ ])
```

מחזירה 0 אם המחרוזות זהות, כלומר שוות בכל תו עד לתו '\0'

אחרת מוחזר מס' שונה מ-0: חיובי (1) אם המחרוזת הראשונה

גדולה יותר , לקסיקוגרפית – כלומר לפי סדר מילוני, סדר המופיע

בטבלת ASSCII , ושלילי (-1) אם היא קטנה יותר.

- פונקציה המעתיקה מחרוזת שניה לראשונה רק מוגבלת בכמות תאים להעתקה הנקבעת ע"י המשתנה n:

```
char strncpy(char str1[ ],char str2[ ],int n)
```

חזרה - הספריה string.h – פונקציות לדוגמא

- פונקציה המבצעת מילוי מחרוזת בתו מסוים:

```
char strset(char str[],char ch);
```

- פונקציה המבצעת מילוי מחרוזת בתו מסוים n פעמים:

```
char strnset(char str1[],char ch,int n);
```

- פונקציה המבצעת שרשור n תווים מ-str2 בסיום str1:

```
char strncat(char str1[],char str2[],int n)
```

הספריה ctype.h פונקציות על תווים

- הספריה ctype.h שימושית כשעובדים עם תווים, הפעולות שלה אינן על מחרוזת שלמה אלא על תו, למשל הפונקציות הבאות מחזירות:

1 אם יש ב- ch יש אות גדולה , 2 קטנה, אחרת 0

```
int isalpha(char ch);
```

מספר בין 1 ל-9 אם ב- ch יש מספר, אחרת 0

```
int isdigit(char ch);
```

מספר בין 1 ל-9 אם ב- ch יש אות קטנה, אחרת 0

```
int islower(char ch);
```

מספר בין 1 ל-9 אם ב- ch יש אות גדולה, אחרת 0

```
int isupper(char ch);
```

חזרה - פונקציות שימושיות על תווים

- הספריה ctype.h שימושית כשעובדים עם תווים, הפעולות שלה אינן על מחרוזת שלמה אלא על תו.
- למשל הפונקציות הבאות מבצעות:

הפיכת אות גדולה לקטנה: `char tolower(char ch);`

הפיכת אות קטנה לגדולה: `char toupper(char ch);`

- אם tolower לא מקבלת אות גדולה אז היא לא משנה אותה, וכך גם אם toupper לא מקבלת אות קטנה.
- true , מצב אמת, משמעו מס' השונה מ-0.

שאלות על השעור הקודם?

נושאי פרק כתובות ומצביעים

- מהן כתובות בזיכרון
- טיפוס משתנה – גודל בזיכרון
- פעולות עם כתובות
- שמירת משתנה בכתובת מסוימת בזיכרון
- טבלת המשתנים
- מהם מצביעים
- מצביעים ופונקציות
- מצביעים ומערכים

כתובות בזיכרון

- טענו כי הזיכרון של המחשב מורכב מתאים, בתים רבים, בכל בית מאוחסנים נתונים.
- במחשבים כיום, נייחים וניידים, מדובר באחסון של מיליונים ואפילו מיליארדים של נתונים.
- לכל תא בזיכרון יש כתובת יחידה (מספר תא).
- כל משתנה נשמר בזיכרון החל מכתובת מסוימת ועד תום האזור שהוגדר עבורו.
- משתנה יכול לתפוס יותר מתא אחד, בהתאם לטיפוס שלו.

כתובות בזיכרון

- למשל, כשמגדירים משתנה על-ידי: int i;
- המחשב היודע מהו הטיפוס המוגדר, במקרה הזה int, מקצה עבור המשתנה i מקום בזיכרון בכתובת שהוא בוחר.
- גודל המקום השמור למשתנה תלוי בטיפוס ועבור int הוא שומר ארבעה בתים כלומר רצף של 32 סיביות.
- בכל מחשב ישנה טבלה של כתובות המשתנים.
- בטבלה מצוינים כל המשתנים, הטיפוסים והכתובות.

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i;
}
```

הזיכרון

זבל

002BFD18

טבלת הכתובות

כתובת	משתנה
002BFD18	i

גם באתחול ערך המשתנה מתעדכן בזיכרון

כתובת כלשהי שהמחשב בחר

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i;
    i=10;
}
```

הזיכרון

10

002BFD18

טבלת הכתובות

כתובת	משתנה
002BFD18	i

גם באתחול ערך המשתנה מתעדכן בזיכרון

כתובת כלשהי שהמחשב בחר

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i=10;
}
```

הזיכרון

10

002BFD18

טבלת הכתובות

כתובת	משתנה
002BFD18	i

גם באתחול ערך המשתנה מתעדכן בזיכרון

כתובת כלשהי שהמחשב בחר

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i=10;
    char ch='A';
}
```

הזיכרון

10

002BFD18

'A'

002DFED8

טבלת הכתובות

משתנה	כתובת
i	002BFD18
ch	002DFED8

כך גם עבור משתנים מסוגים אחרים

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i=10;
    char ch='A';
}
```

הזיכרון

10

002BFD18

65

002DFED8

טבלת הכתובות

משתנה	כתובת
i	002BFD18
ch	002DFED8

זכור, למעשה נשמר ערך האסקי של התו

כתובות בזיכרון – גודל המשתנה

- בטבלת הכתובות נשמר נתון נוסף, שהוא מספר התאים (בתים) שמשמשים לייצוג המשתנה המוגדר.
- הטיפוס **char** מיוצג על-ידי בית אחד בלבד.
- הטיפוס **int** מיוצג על-ידי ארבעה בתים.
- הטיפוסים **double** ו- **float** מיוצגים ע"י שמונה בתים (ברוב הקומפיילרים).
- כשמתייחסים בתוכנית לשם של משתנה, המחשב בודק בטבלה איפה המשתנה נמצא בזיכרון וכמה מקום הוא תופס, וכך הוא יודע להביא לנו את הערך שלו.

כתובות בזיכרון - דוגמא

התוכנית

```
int main()
{
    int i=10;
    char ch='A';
}
```

הזיכרון

0	0	0	10

002BFD18

65

002DFED8

טבלת הכתובות

משתנה	כתובת	גודל
i	002BFD18	4
ch	002DFED8	1

לטיפוס int המחשב מקצה 4 תאים (4 בתים), המכילים 32 סיביות החל מהכתובת הזו.

לטיפוס char המחשב מקצה תא (בית אחד) המכיל 8 סיביות החל מהכתובת הזו.

שאלות?

כתובות – איך מתייחסים אליהן

- שפת C מאפשרת לנו לדעת מהי הכתובת בזיכרון שבה נשמר משתנה מסוים.
- מיקום הכתובת בזיכרון מאפשר גישה ישירה לכתובות בזיכרון דבר המייעל את עבודת המחשב.
- הפעולה & (shift+7) נותנת את הכתובת של המשתנה.
  - למשל &i היא הכתובת בזיכרון של המשתנה i.
  - בדוגמא הקודמת הפעולה &i החזירה לנו את הכתובת 002BFD18 שהיא הכתובת שהחל ממנה נמצא הזיכרון של המשתנה i.

כתובות – איך ניגשים אליהן

- הפעולה & נותנת את הכתובת של משתנה הצמוד לה.
  - למשל &i היא הכתובת בזיכרון של המשתנה i.
  - בדוגמא הקודמת &i היה נותן 002BFD18 .
- הפעולה * ניגשת לערך שנמצא בכתובת מסוימת.
  - אם נכתוב *(002BFD18) נגיע לערך שנמצא בכתובת 002BFD18 שהיא כתובת של המשתנה i.
  - גם אם נכתוב *(&i) נגיע לערך של המשתנה i (כי בפועל זהו הערך המספרי הנמצא בכתובת של i).

כתובות – מה אפשר לעשות איתן

- לא ניתן לשנות כתובת של משתנה.
- כלומר לא ניתן לכתוב: ~~&i= 002BFD18~~ כי כך ניתן לשנות את הערך שנמצא בכתובת מסוימת וזה אסור!!!

אם נכתוב i=10; זה בדיוק כמו לכתוב *(&i)=10;

אם נכתוב ch='A'; זה בדיוק כמו לכתוב *(&ch)='A';

- המצביע מצביע על תוכן הנמצא בכתובת מסוימת, הכתובת מסומנת ע"י & והתוכן ע"י *.

כתובות – מה אפשר לעשות איתן

- מסוכן ועשוי לגרום נזק שינוי ערך שנמצא בכתובת כלשהי שאיננה כתובת של משתנה שהוגדר ע"י המתכנת.
- לדוגמה אם נכתוב *(002BFD18) = 12569

אנחנו עשויים לשנות כתובת של מקום בזיכרון ששייך למערכת ההפעלה, ולגרום לתוכנית לעוף או לחילופין לנזקים אחרים הקשורים בכתובות בזיכרון המחשב.

- לכן נשנה רק ערכים בכתובת של משתנים שהגדרנו.

*(&i)=123456;

בשביל מה כל זה טוב?

- כדי לשנות ערך של משתנה אפשר פשוט לשים בו ערך.
- למשל ניתן לבצע את ההשמה הבאה: i=10
- אם כך, אז למה צריך לדעת מה הכתובת של אותו משתנה?
- מדוע נבצע השמה לערך של משתנה ישירות לכתובת שלו ולא להשתמש פשוט בשם המשתנה?
- אם כך מהו ההבדל בין גישה למשתנה דרך שמו לבין גישה אליו דרך כתובתו?

בשביל מה נשתמש בכתובות

- ידוע כי משתנה מוכר רק בתוך הפונקציה שבה הוא הוגדר, למעט משתנה גלובלי המוכר בכל התוכנית .
- נשתמש בכתובות של משתנים כדי לשנות את ערכם מתוך פונקציה אחרת, שבה הם לא הוגדרו.
- אם רוצים שפונקציה תשנה מספר ערכים של מספר משתנים שונים (ולא רק תחזיר ערך אחד), אז נעביר לפונקציה את הכתובות שלהם.
- שינוי ערכו של משתנה דרך כניסה לכתובתו מאפשר לפונקציה לשנות באופן ישיר את ערכו בזיכרון.

דוגמא שהכרנו

- השתמשנו בהעברת כתובת בפונקציה scanf( ), שמקבלת את כתובת המשתנה שאליו יוכנס הקלט.
- למשל:

```
scanf("%d", &grade);
```

- המשתנה grade אינו מוגדר בפונקציה scanf( ) והיא גם לא מחזירה לתוכו ערך אבל היא מבצעת השמה של התוכן שנקלט מהמסך ישירות לכתובת שלו, של grade ע"י השימוש ב- &grade.
- הפונקציה scanf( ) מקבלת את הכתובת של המשתנה בזיכרון באמצעות סימן ה- & לפני המשתנה.

דוגמא שהכרנו

- סימון זה (&) מאפשר לה לכתוב לתוכו את הקלט ישירות לתוך כתובת המשתנה. (זה ניתן לביצוע באמצעות הוספת ה- * לפני המשתנה או לפני כתובתו)
- קידוד הקלט (למשל ע"י הסימן: "%d") מיידע את פונקציית הקליטה: scanf(); מה גודל המשתנה בזיכרון, כלומר לכמה תאים (בתים) הקלט אמור להיכנס.
- %c – בית אחד בזיכרון – שמונה סיביות.
- %d – ארבעה בתים בזיכרון – 32 סיביות.
- %f – שמונה בתים בזיכרון – 64 סיביות.

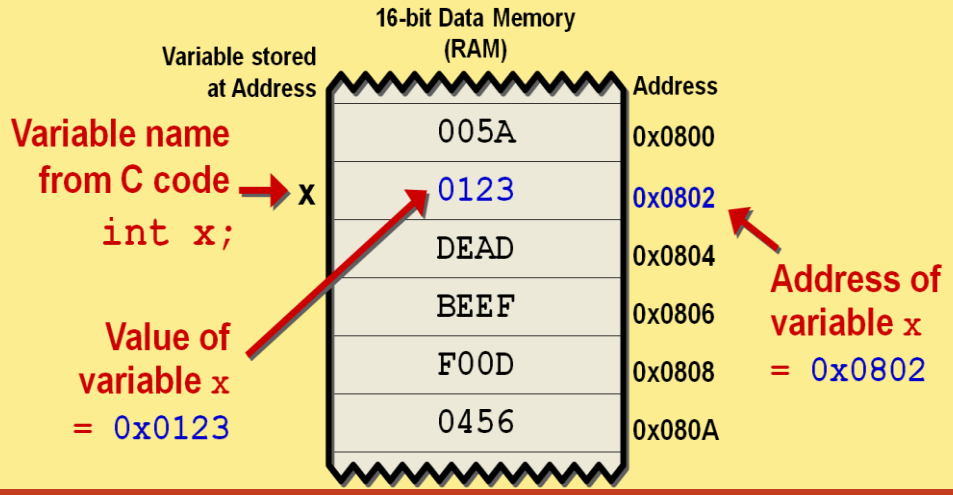
כתובות בזיכרון – סיכום פעולות

- מציאת הכתובת באמצעות: &i
- גישה למשתנה לפי הכתובת עם הפעולה: *
- צורות הכתיבה הנ"ל מזמנות לנו אפשרות לשנות כמה ערכים של משתנים בפונקציה.
- כל כתיבה ישירה בזיכרון ללא שימוש בשם המשתנה היא מהירה יותר ומבצעת עדכון של ערך המשתנה באופן מהיר ומדויק יותר.

עבודה עם כתובות - מצביעים

- ב-C קיימים סוגי משתנים לשמירת כתובות בזיכרון.
- הם נקראים "מצביעים" או "פויינטרים".
- באופן כללי, כדי להגדיר משתנה מסוג מצביע, יש לרשום את סוג המשתנה שעליו מצביעים, ולהוסיף את סימן ה- * לפני שם המשתנה.

```
double *ptr_double;
float *ptr_float;
int *ptr_int;
char *ptr_char;
```



מצביעים וכתובות – נקודות לתשומת-לב

- יש לשים לב שלקבועים וביטויים אין כתובת.
- למשל, לא נכתוב:

&5

&(i*2)

- אך ורק למשתנים יש כתובת.
- כיוון שלא ניתן לשנות כתובת של משתנה (כיוון שהמחשב קובע אותה באופן אוטומטי).
- לא ניתן לבצע על כתובת השמה: ~~&a=&~~b

מצביעים – שתי נקודות נוספות

- במידה ויש צורך לסמן שמצביע לא מצביע למשתנה (למשל לפני שהתחלנו להשתמש בו), אזי מקובל לתת לו את הערך NULL.
- למשל:

```
int *my_pointer_to_int=NULL;
```

NULL הוא קבוע שערכו 0 שמוגדר ע"י #define בספריה stdlib.h.

- קידוד ההדפסה ב- printf( ) להדפסת כתובת בזיכרון הוא "%p", למשל להדפסת כתובת המשתנה i נכתוב:

```
printf("%p", &i);
```

מצביעים – סיכום ביניים

- אלה משתנים שמשמשים לשמירת כתובת של משתנה אחר.
- סוג המצביע נרשם בהגדרה ע"י סוג המשתנה שמצביעים עליו ו-*.
- למשל:

```
int mis=0;
int *my_ptr=NULL;            //int *my_ptr=&mis;
my_ptr = &mis;
```

- אפשר לגשת למשתנה שהפויינטר מצביע עליו בעזרת *. למשל המשמעות של mis:

mis=5; שקול ל- *my_ptr=5;

*my_ptr=75; שקול ל- mis = 75

שאלות?

מצביעים – דוגמא

- נתונה בעיה:

יש לקלוט שני ערכים מספרים לשני משתנים שונים ולבצע החלפה ביניהם ע"י שימוש בפונקציה כך בסיום הפונקציה הערך שהיה במשתנה הראשון יהיה במשתנה השני ולהיפך.

- ראשית, נכתוב פונקציה שמקבלת כתובות של אותם שני משתנים מסוג int, ומחליפה בין הערכים שלהם תוך כדי שימוש במצביעים.
- ע"י השימוש במצביעים ההחלפה תתבצע בין הערכים שבכתובת ובכך יוחלפו הערכים האמיתיים שבתוכנית.

מצביעים – דוגמא

- יש לזכור כי עד עכשיו לא יכולנו לכתוב פונקציה להחלפת ערכים של שני משתנים מסוג int, כי שינוי משתנים מסוג int בפונקציה לא משפיע על ערכם המקורי, ופונקציה יכולה להחזיר רק ערך אחד.
- כעת כשיש לנו אפשרות להשתמש בכתובות המשתנים שבזיכרון, כלומר במצביעים נוכל לבצע את המשימה באמצעות פונקציית swap() פשוטה.
- לסיכום, השימוש במצביעים מאפשר העברת נתונים ושינוי ערכי משתנים בפונקציות ללא צורך בביצוע החזרה של ערכים מהפונקציה לתוכנית הראשית.

מצביעים – דוגמא


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
```

ערך המשתנה שכתובתו first יוחלף עם ערך המשתנה שכתובתו second, ההחלפה באמצעות תא העזר temp

- השימוש יבוצע על-ידי הפקודה:

```
swap(&mispar1, &mispar2);
```

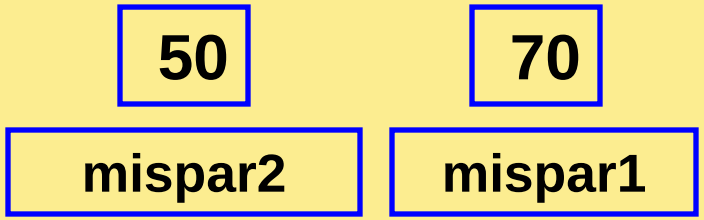
כתובת המשתנה הראשון כתובת המשתנה השני

דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```

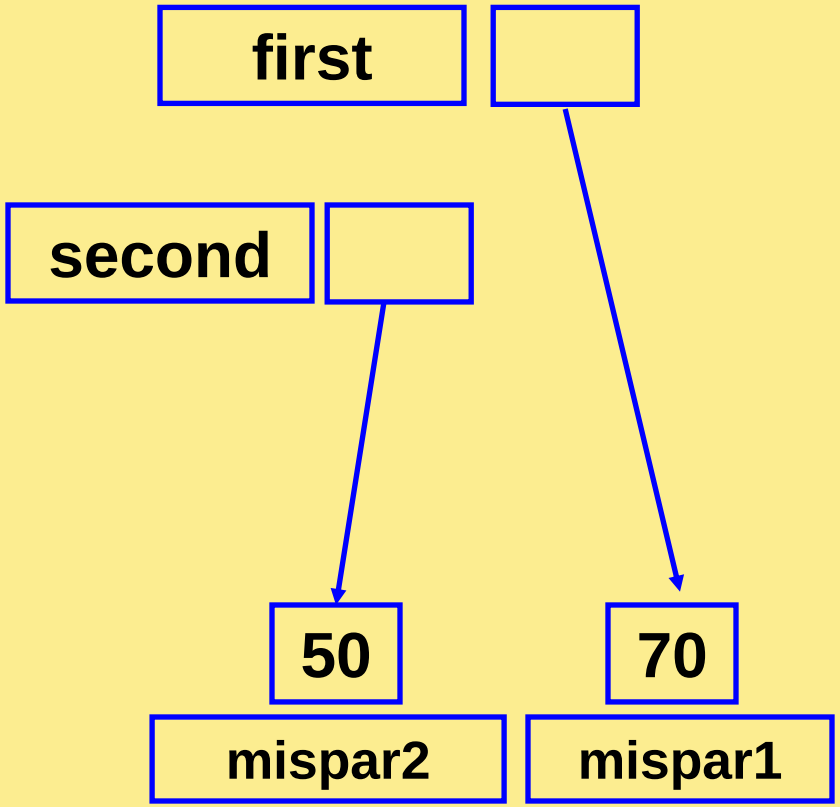
דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



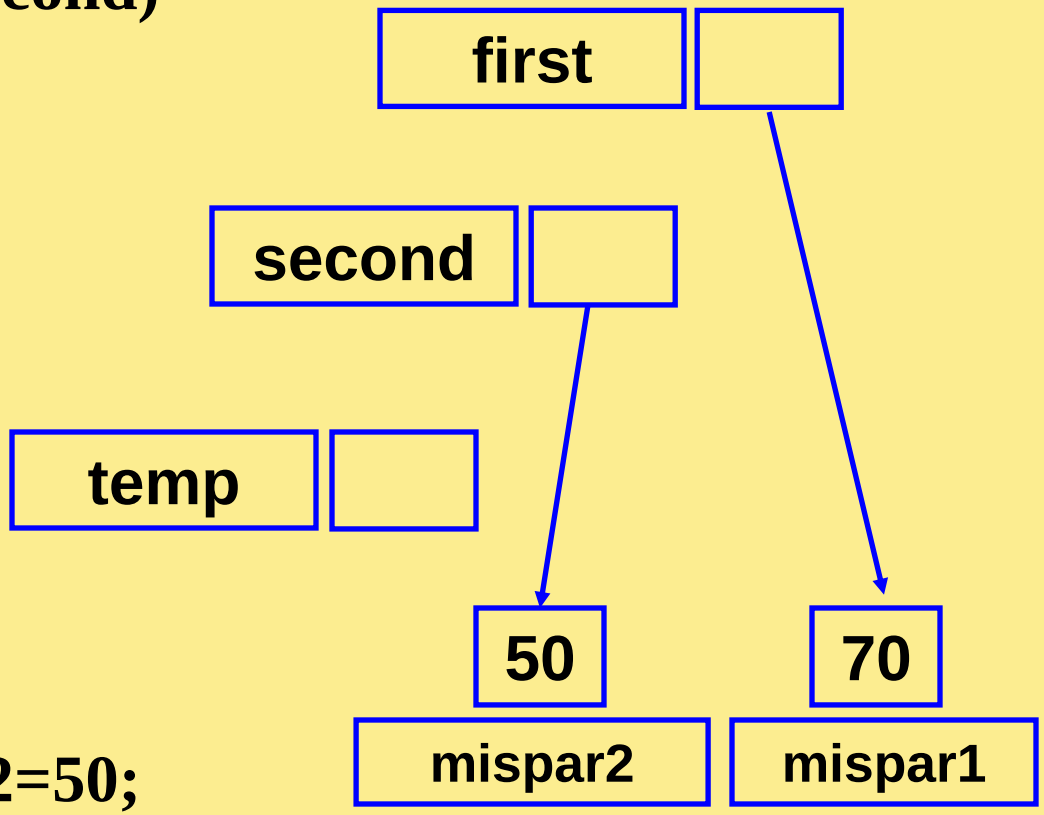
דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



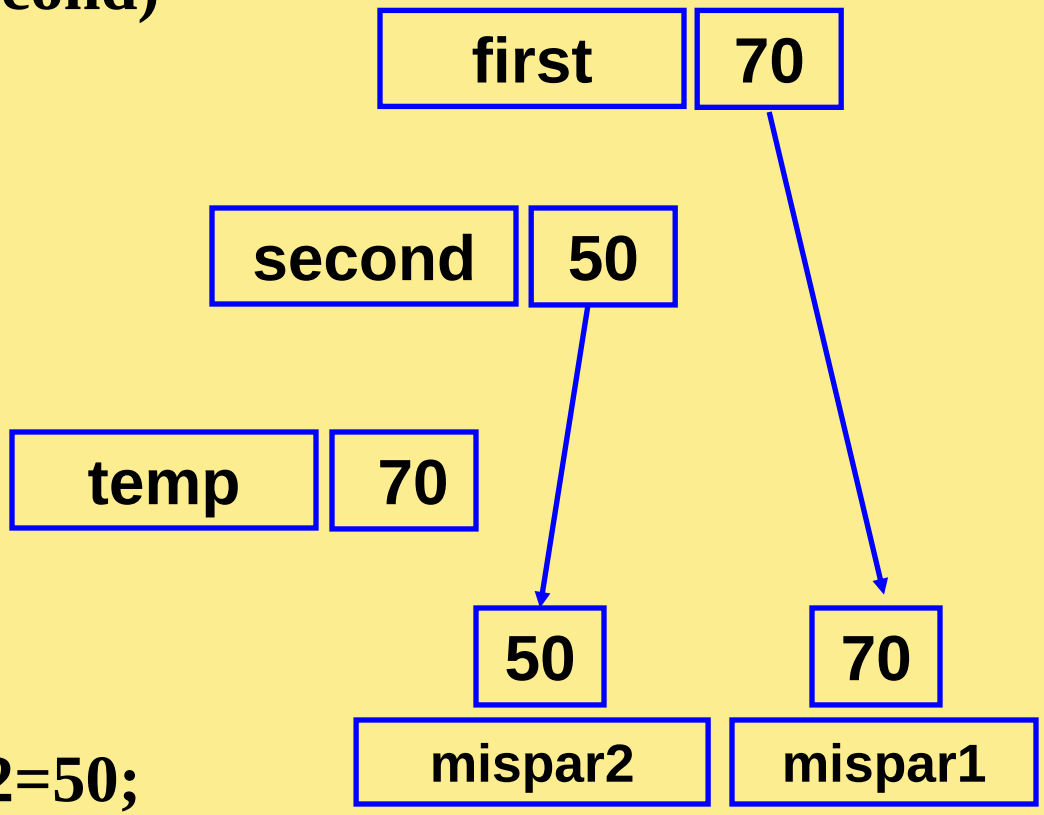
דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



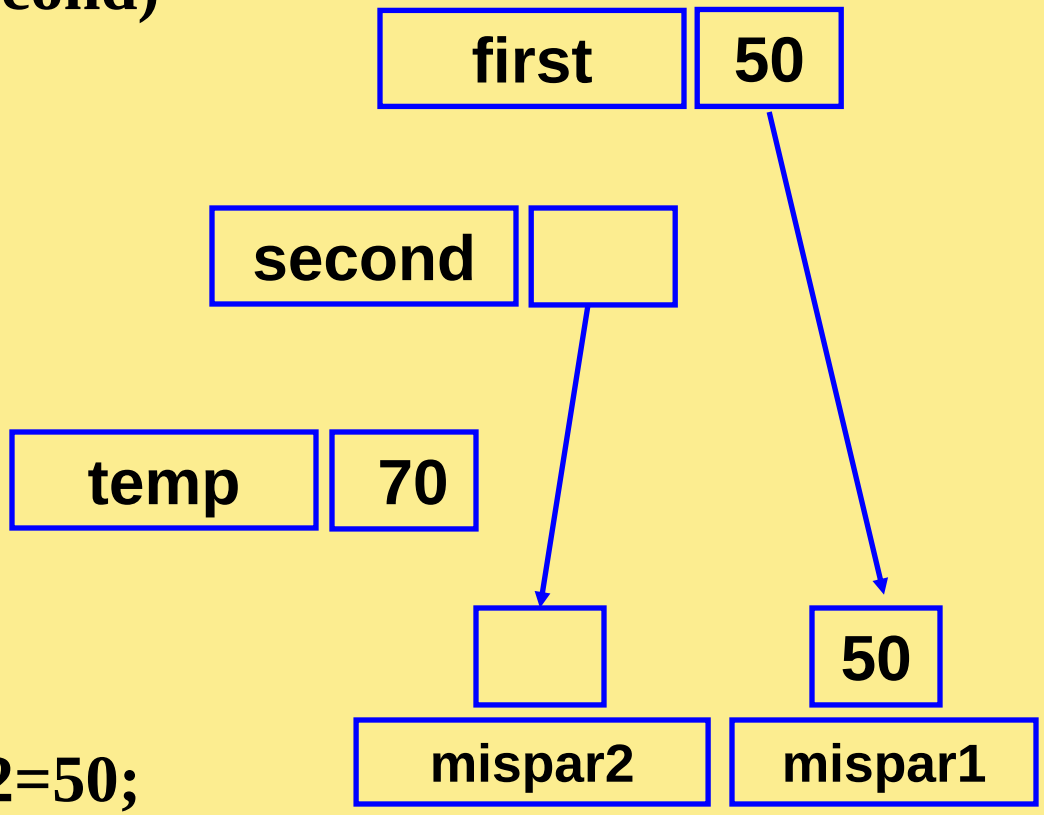
דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



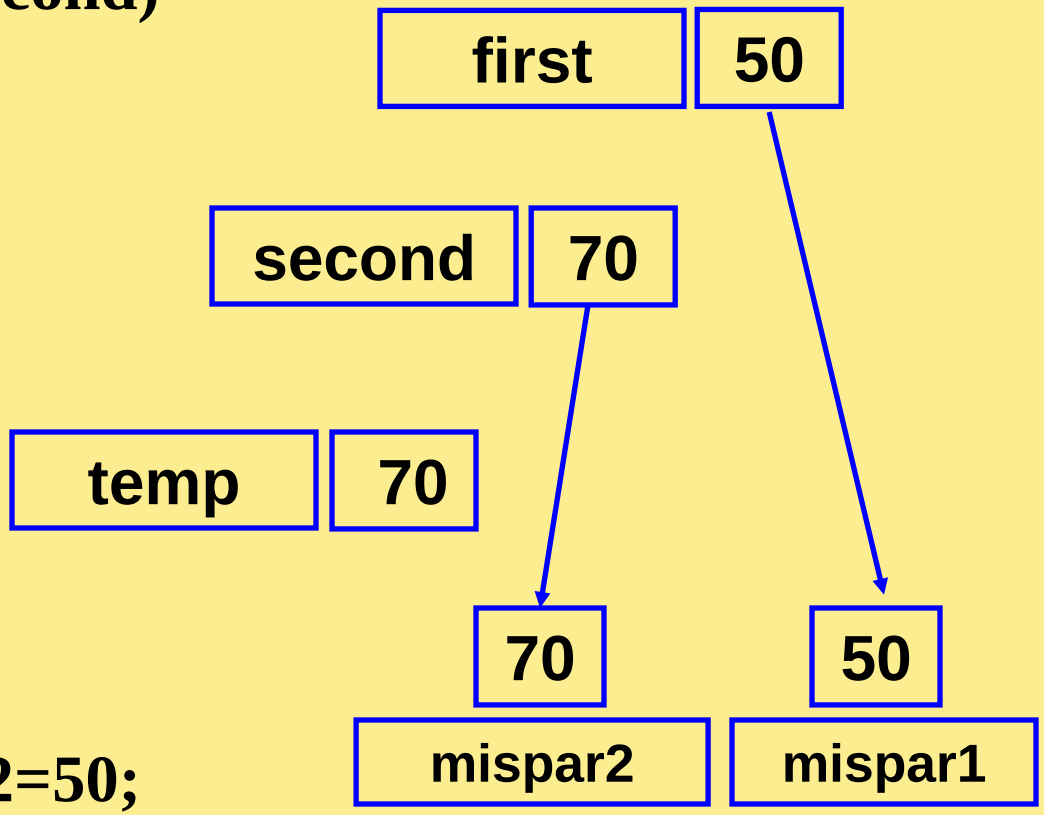
דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



דוגמא: פונקציה להחלפה בין ערכי משתנים


```
void swap(int *first, int *second)
{
    int temp;
    temp=*first;
    *first=*second;
    *second=temp;
}
int main()
{
     int mispar1=70, mispar2=50;
     swap(&mispar1,&mispar2);
}
```



דוגמא: פונקציה להחלפה בין ערכי משתנים


```
#include<stdio.h>
void swap(int *first, int *second)
{
    int temp;
    printf("\nSwap function");
    printf("\nBEFORE SWAP:--->address first=%p  address
    second=%p\n", first, second);
    printf("BEFORE SWAP:--->first=%d
    second=%d\n",*first,*second);
    temp=*first;        *first=*second;        *second=temp;
    printf("\nAFTER SWAP:--->address first=%p  address
    second=%p\n", first, second);
    printf("AFTER SWAP:--->first=%d
    second=%d\n",*first,*second);
}
```

דוגמא: פונקציה להחלפה בין ערכי משתנים


```
int main()
{
    int mispar1, mispar2;
    printf("Please enter two int numbers for swap:\n");
    scanf("%d %d",&mispar1,&mispar2);
    swap(&mispar1,&mispar2);
    printf("\nMain program\n");
    printf("mispar1=%d  mispar2=%d\n",mispar1,mispar2);
    printf("address mispar1=%p  address
    mispar2=%p\n",&mispar1,&mispar2);
}
```

עוד על פונקציות ומצביעים

- נציין שהערך המוחזר מפונקציה יכול להיות גם מצביע, למשל:

```
int *address_of _the_higher_number(int *mis1, int *mis2);
```

- אבל החזרת כתובת משתנה שהוגדר בתוך הפונקציה תגרום לשגיאה. הסיבה לכך היא שהמשתנים המוגדרים בתוך הפונקציה נעלמים עם סיומה.
- למשל הפונקציה הבאה תגרום לשגיאה:

```
int *give_pointer_to_zero()
{
   int i=0;
   return &i;
}
```

עוד על פונקציות ומצביעים

- ניתן לבצע הגדרה של פונקציית מצביע המכילה משתנה מסוג מצביע מהטיפוס של הפונקציה ולהחזיר את כתובתו לתוכנית הראשית, לדוגמה:

```
#include <stdio.h>
int *funcsum(int mis1, int mis2)
{
        int sum=NULL;
        sum = mis1 + mis2;
        return &sum;
}
void main()
{
        int num1 = 10, num2 = 5, *sum = funcsum(num1, num2);
        printf("Sum=%d\n",*sum);
}
```

שאלות?

שליחת ערכים by address/by value

- שליחת ערכים ע"י שימוש ב- by value:
  - הערך מועתק לפרמטר מקומי של הפונקציה.
  - נעשה בו שימוש בתוך הפונקציה בלבד
  - כשהפונקציה מסתיימת הפרמטר המקומי נמחק
- שליחת ערכים ע"י שימוש ב- by address:
  - הערך לא מועתק לפרמטר המקומי של הפונקציה אלא כתובתו מועברת לתוך הפרמטר המקומי.
  - נעשה בו שימוש בתוך הפונקציה כאשר כתובתו משתמרת
  - כיוון שרק הכתובת של הערך מועברת (משתנה מקומי מסוג פוינטר מאחסן את הכתובת) ולכן האופרטור * מאפשר גישה לערך עצמו.

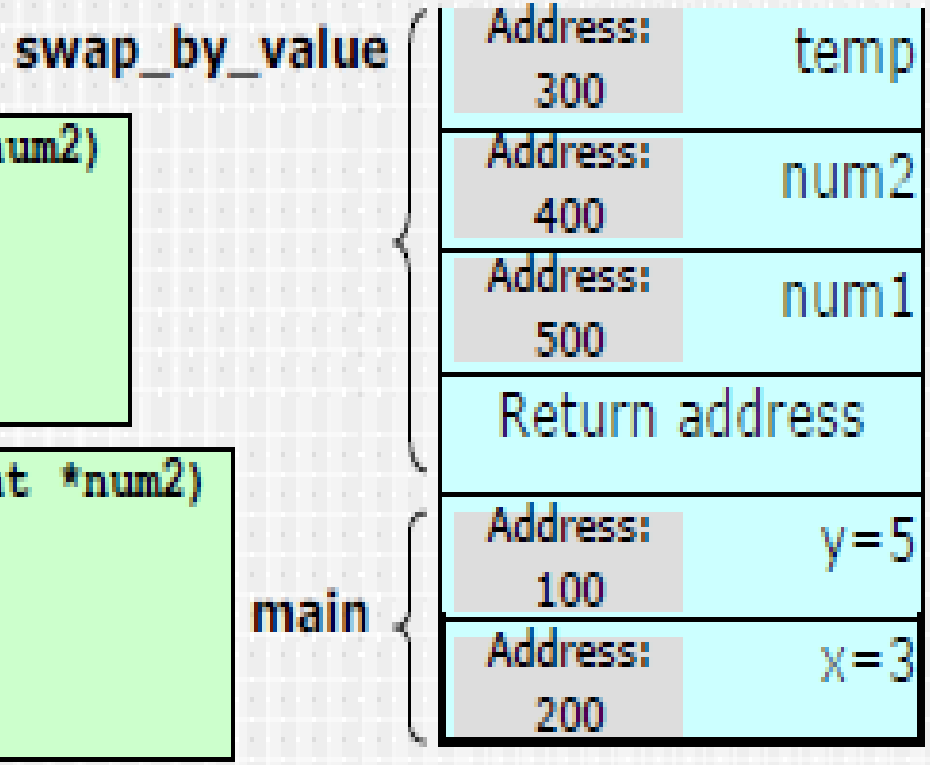
שליחת ערכים by value- דוגמא


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

swap מקבלת את x ואת y ומחליפה ביניהם.



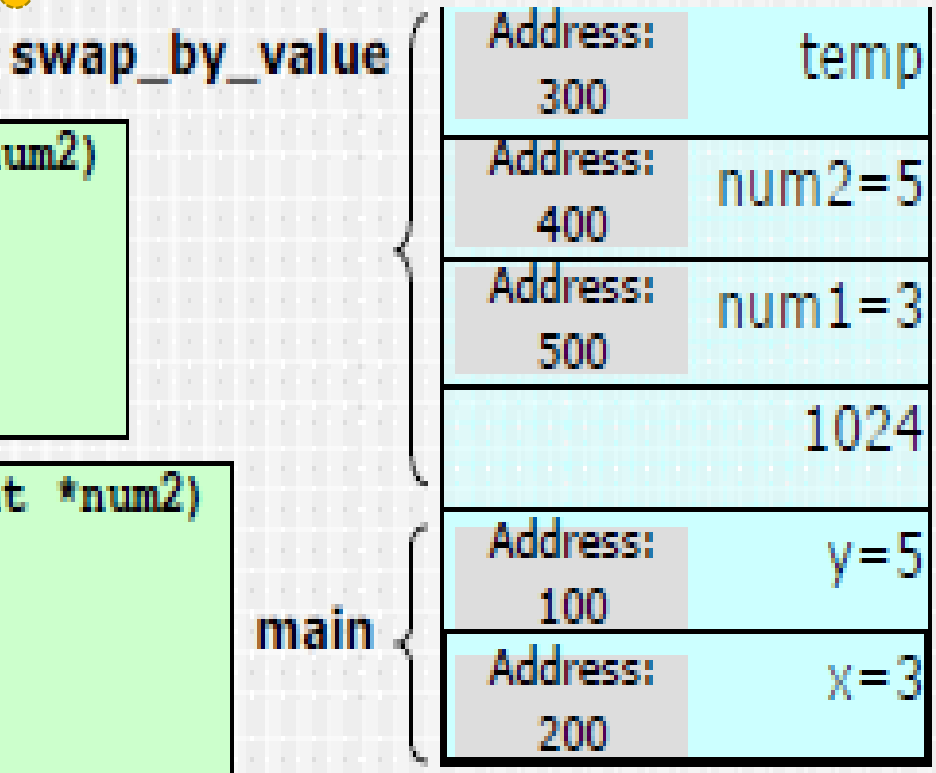
שליחת ערכים by value- דוגמא (2)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

העברת פרמטרים



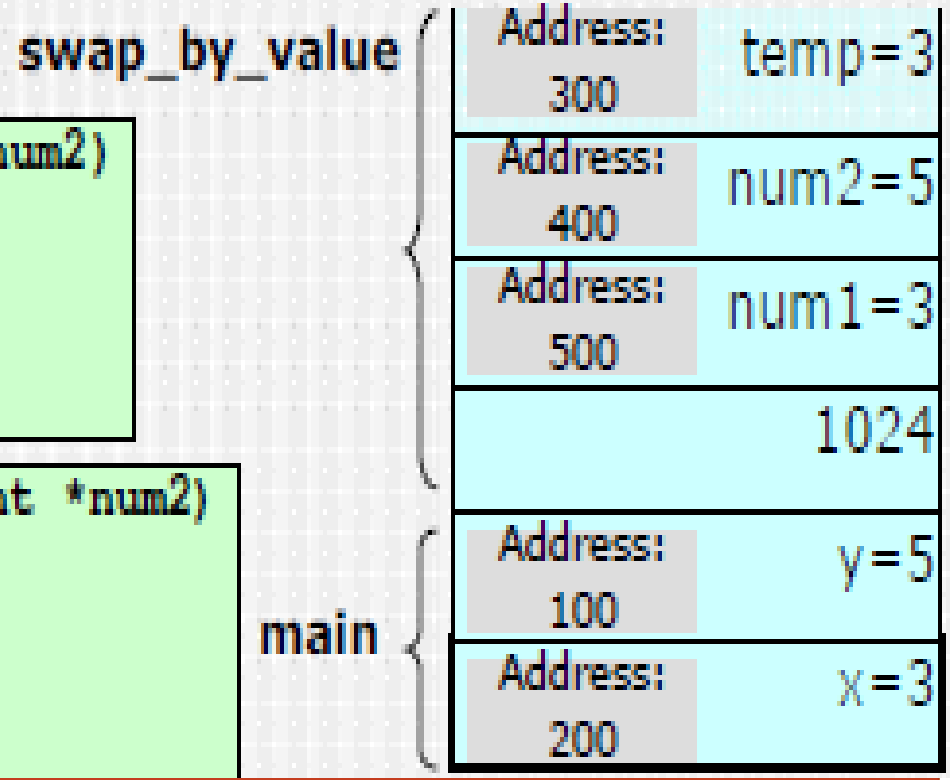
שליחת ערכים by value- דוגמא (3)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

temp=num1;



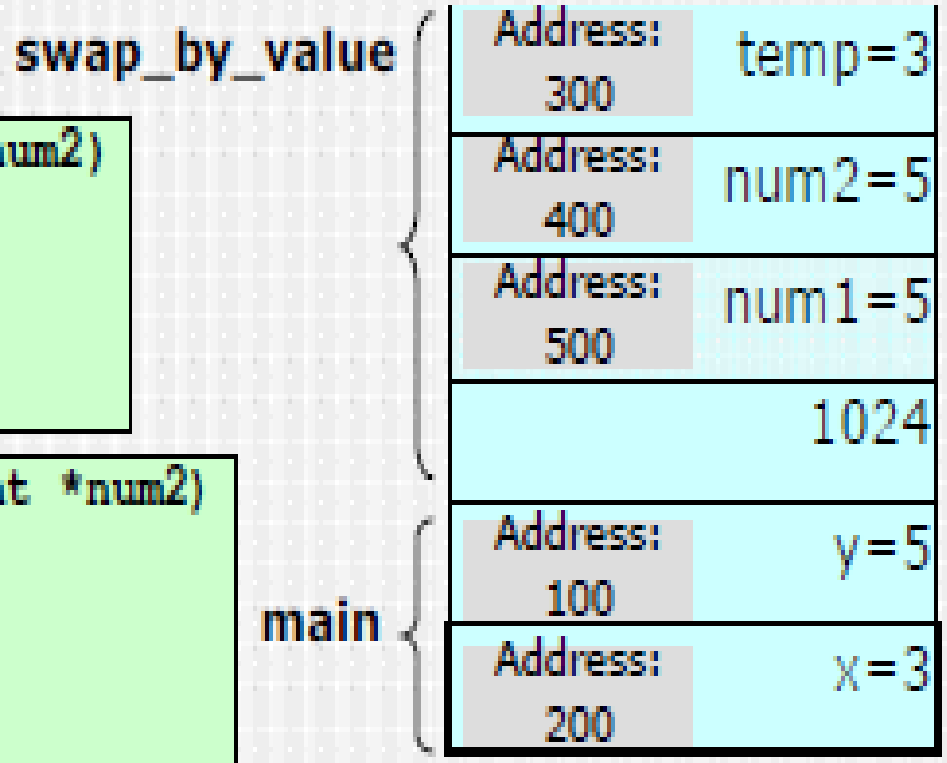
שליחת ערכים by value- דוגמא (4)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

num1=num2;



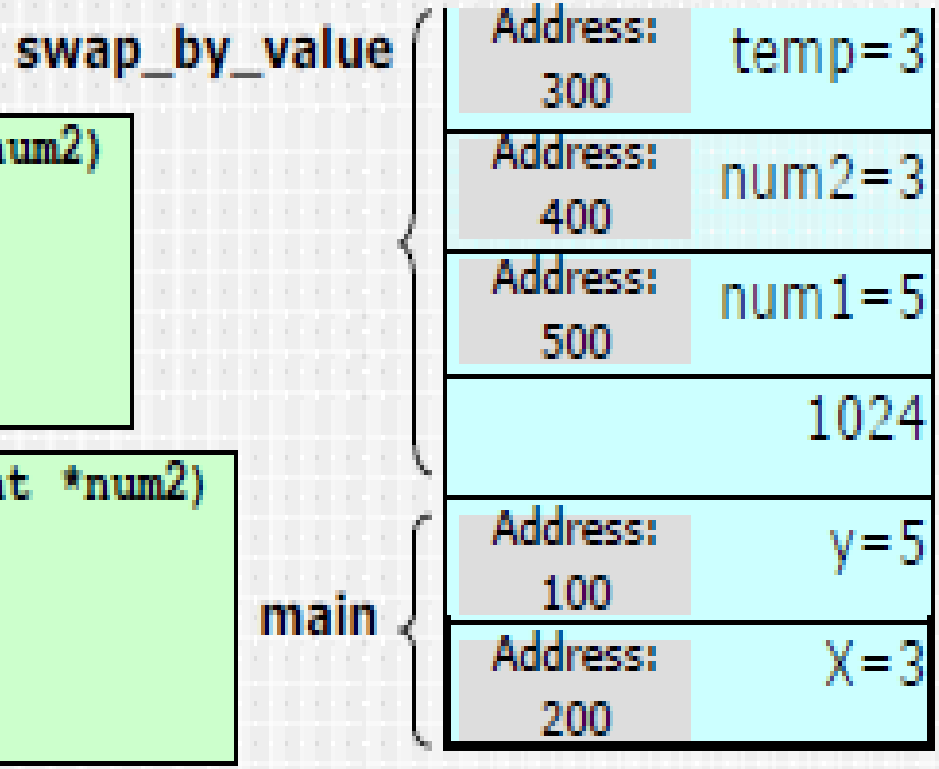
שליחת ערכים by value- דוגמא (5)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

num2 = temp;



שליחת ערכים by value- דוגמא (6)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

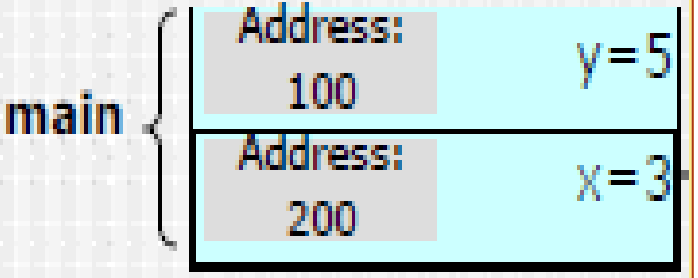
לאחר סיום הפונקציה

swap_by_value

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

שימו לב – x ו-y לא
השתנו בכלל!

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```



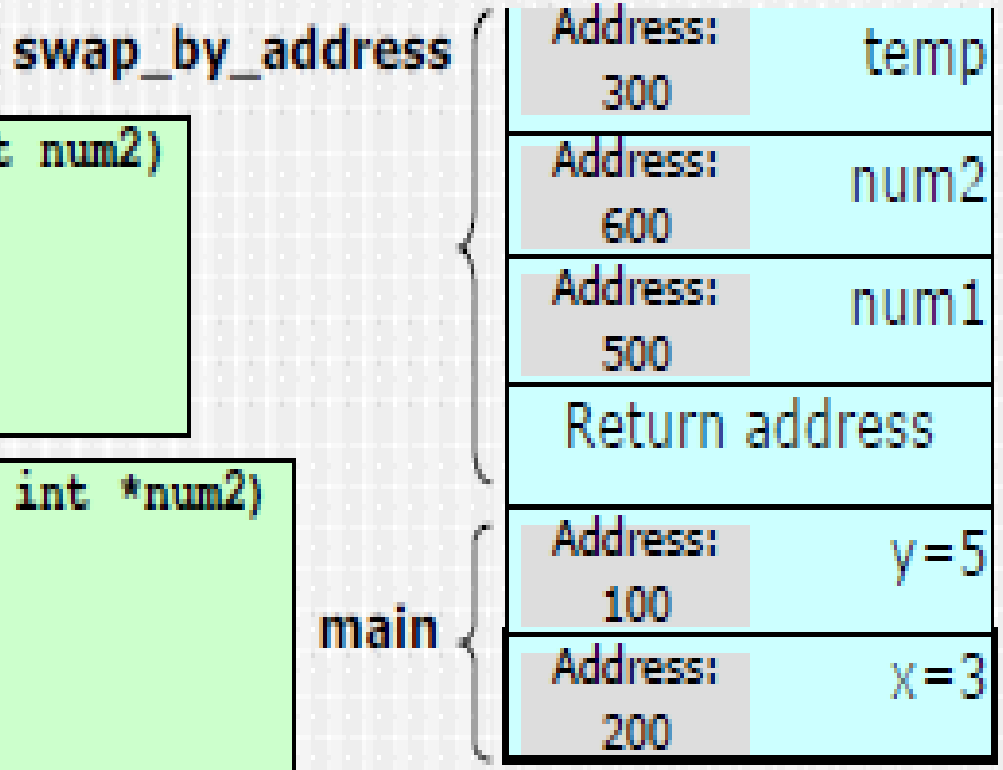
שליחת ערכים by address- דוגמא (1)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

swap מקבלת את x ואת y ומחליפה ביניהם.



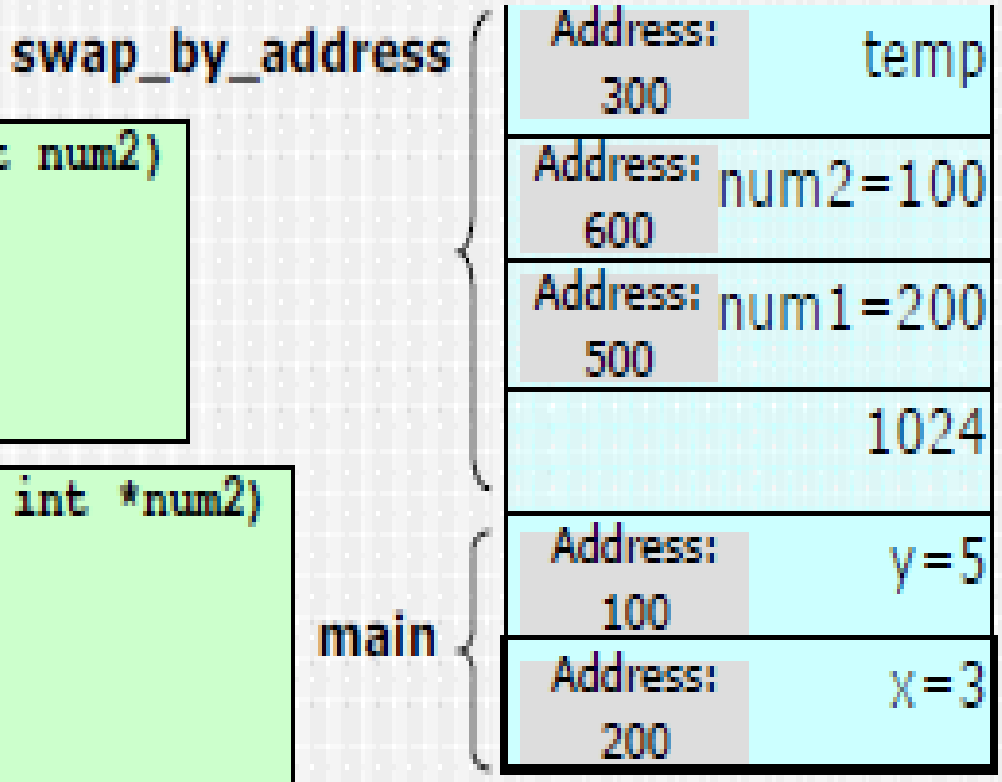
שליחת ערכים by address - דוגמא (2)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

העברת פרמטרים



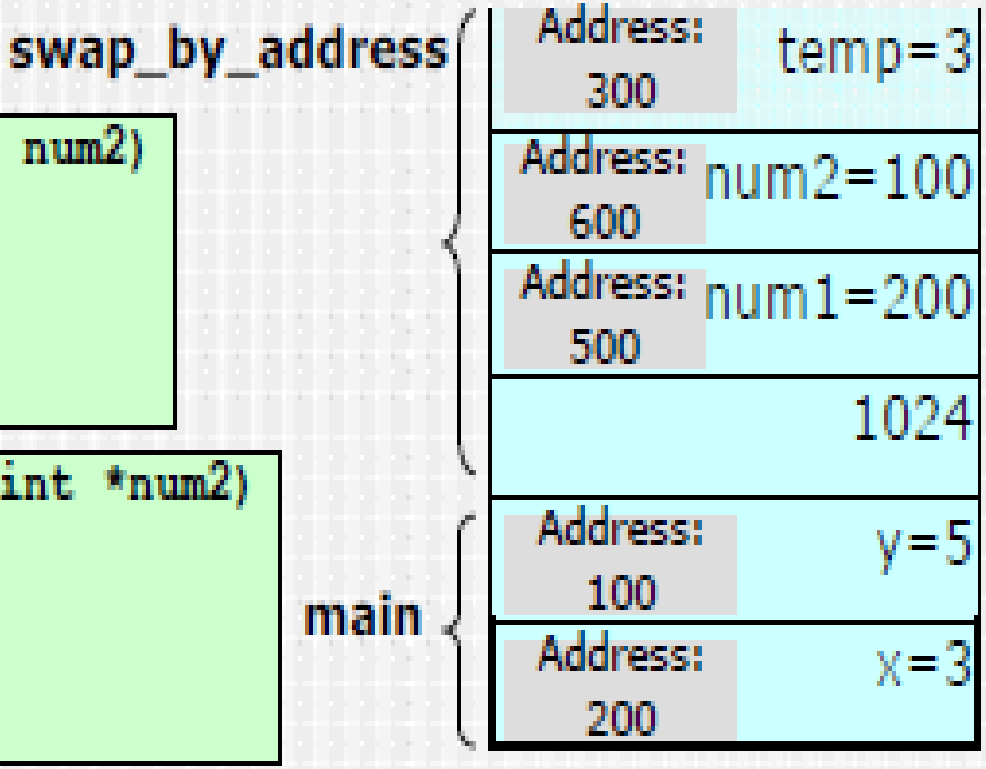
שליחת ערכים by address - דוגמא (3)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

temp = *num1



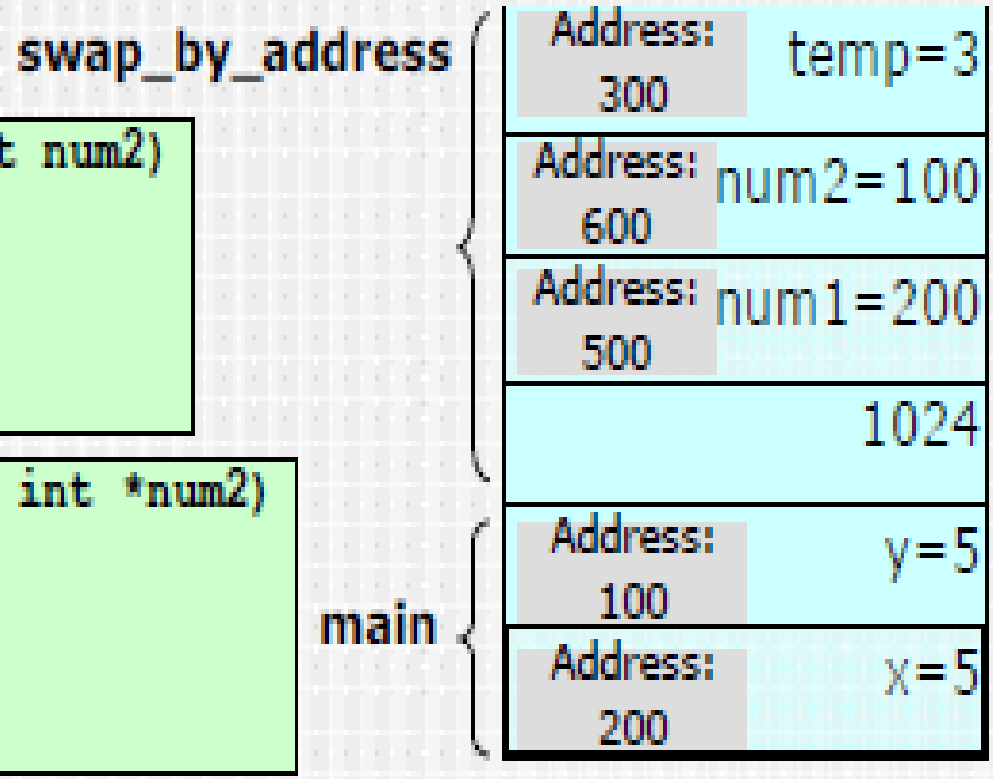
שליחת ערכים by address - דוגמא (4)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

***num1 = *num2**



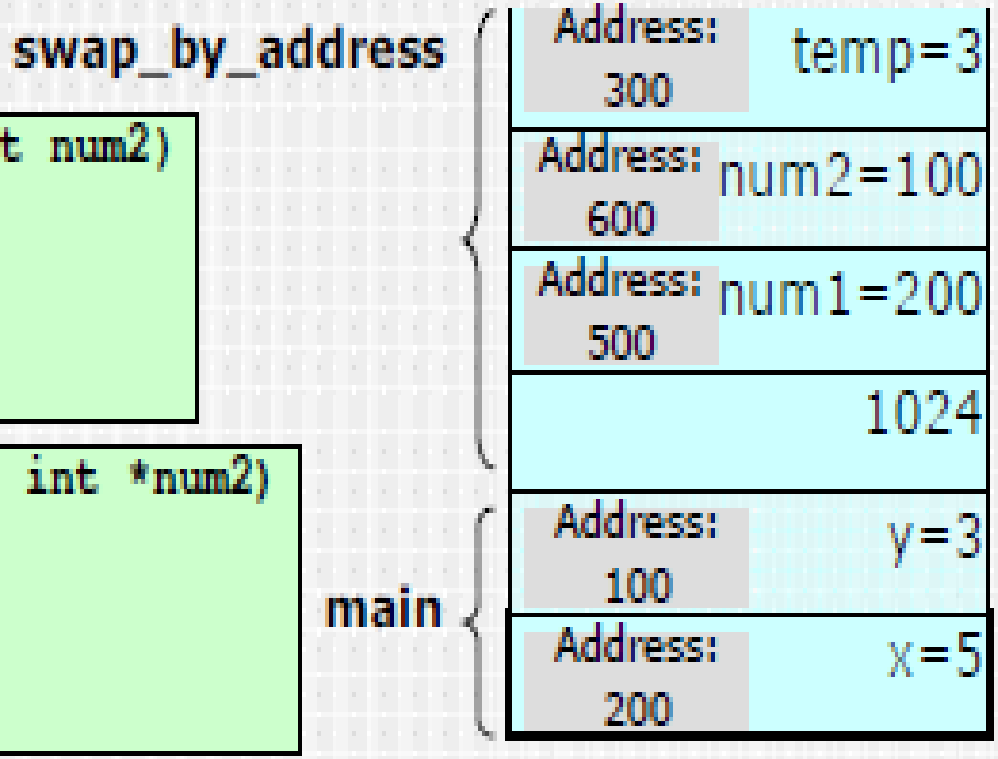
שליחת ערכים by address - דוגמא (5)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

***num2 = temp**



שליחת ערכים by address - דוגמא (6)


```
void main()
{
    int x=3, y=5;
    swap_by_value(x, y);
    swap_by_address(&x, &y);
}
```

swap מקבלת את x ואת y ומחליפה ביניהם.

```
void swap_by_value(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

שימו לב – x ו-y השתנו! וכעת ערכים הוחלפו!

```
void swap_by_address(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

העברת ערכים – טבלת סיכום

	By address (pointer)	By value
הצהרה	func(type name[]) func(type* name)	func(type name)
Scope	בתוך הפונקציה	בתוך הפונקציה
Lifetime	כתובת: רק בתוך הפונקציה. הערכים עצמם: משך החיים המקורי.	בתוך הפונקציה
אפשרות שינוי של הפרמטרים	כתובת: לא הערכים עצמם: כן.	לא

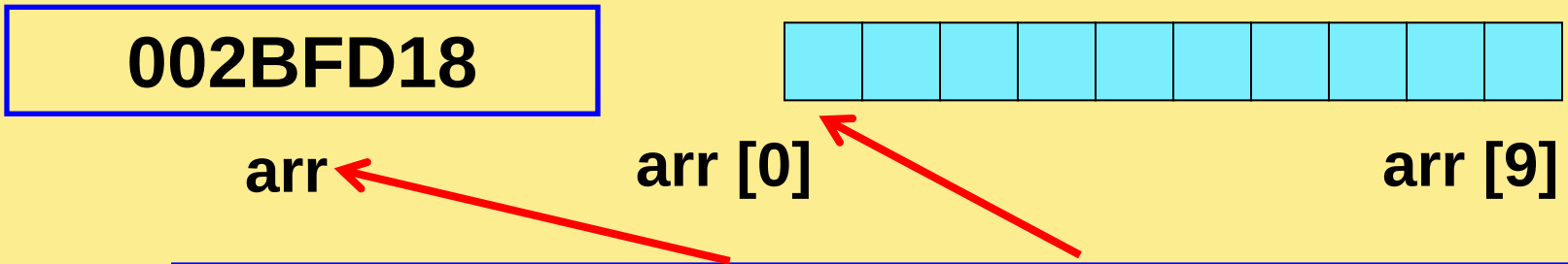
שאלות?

מצביעים ומערכים

- כשכותבים:

```
char arr [SIZE];
```

- המחשב מקצה בזיכרון 10 תאים רצופים עבור 10 תווים, והמשתנה arr (שם מערך) מכיל את כתובת התא הראשון[0].



זו הכתובת של התחלת המערך כלומר של תא arr[0]

ניתן היה גם לכתוב זאת כך: &arr[0] וזה שקול לחלוטין

מצביעים ומערכים

- אי-אפשר לשנות כתובת של מערך (היא נקבעת כשהוא מוגדר), לכן לא ניתן לכתוב:

~~arr=arr1~~

~~arr=0~~

מציבים במצביע את כתובת התא הראשון במערך

- אפשר לשים כתובת של מערך בתוך מצביע:

```
char *arr_ptr=NULL;
arr_ptr=arr;
```

השורה האחרונה שקולה לשורה:

```
arr_ptr = &arr [0];
```

גישה לתאי-מערך לפי הכתובת

- משתנה המציין מערך הוא סוג של מצביע אשר מצביע על התחלת מבנה נתונים בשם מערך (אבל לא להיפך).
- מערך הוא למעשה מצביע שמקבל כתובת בהגדרה ולא ניתן לשנות אותה.
- אם הגדרנו למשל:

```
int arr [SIZE];
int *arr_ptr=NULL;
arr_ptr=arr;
```

(מצביע על התא הראשון במערך)

```
//int *arr_ptr=arr;
```

- כעת ניתן לגשת לתאי-המערך גם לפי הכתובת שלהם, כמו משתנים אחרים.

גישה לתאי-מערך לפי הכתובת

- 3 הפעולות הבאות עושות אותו דבר (מבצעות השמה של המספר 100 בתא מס' 5 במערך):

גישה לתא בכתובת 5 במערך בשלוש שיטות:

```
arr [5]=100;

*(arr+5)=100;

*(arr_ptr+5)=100;
```

- המחשב יודע כמה בתים להתקדם כשאנחנו מבקשים להתקדם ב-5 תאים קדימה, כי הגדרנו שטיפוס הערכים הוא int והוא יודע כמה בתים כל int תופס (כאמור, ערכי המערך שמורים בזיכרון ברצף).

גישה לתאי-מערך לפי הכתובת

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	1	2	3	4	5	6	7	8	9
20CF0	20CF4	20CF8	20CFC	20D00	20D04	20D08	20D0C	20D10	20D14

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
–0	–1	–2	–3	–4	–5	–6	–7	–8	–9
20D20	20D24	20D28	20D2C	20D30	20D34	20D38	20D3C	20D40	20D44

b	a
20D20	20CF0
FFBEED24	FFBEED28

Arrangement of arrays a[ ] and b[ ] inside the memory. Indicated are: external names, values, addresses.

גישה למערך לפי כתובת – דוגמא נוספת

- עבור הדפסת מערך, 3 האפשרויות הבאות עושות בדיוק אותו דבר:

```
int i,arr[SIZE]={12,36,45,78,95,42,64,31,75,19};
int *ptr=NULL;
for (i=0; i<SIZE; i++)
    printf("%4d", arr[i]);
for (i=0; i<SIZE; i++)
    printf("%4d", *(arr+i));
for (ptr=arr; ptr <= &arr[SIZE-1]; ptr++)
    printf("%4d", *ptr);
```

- הדפסה על-ידי גישה רגילה לתאי המערך
- הדפסה על-ידי גישה לכל תא לפי הכתובת שלו יחסית לתא הראשון
- הדפסה ע"י קידום מצביע מכתובת התא הראשון עד כתובת התא האחרון

גישה למערך לפי כתובת – דוגמא נוספת

- עבור הדפסת מערך, 3 האפשרויות הבאות עושות בדיוק אותו דבר:

```
int i;   char *ptr=NULL, arr[SIZE]="Shalom Taly";
```

- הדפסה על-ידי גישה רגילה לתאי המערך

```
for (i=0; i<SIZE; i++)
    printf("%2c", arr[i]);
```

- הדפסה על-ידי גישה לכל תא לפי הכתובת שלו יחסית לתא הראשון

```
for (i=0; i<SIZE; i++)
    printf("%2c", *(arr+i));
```

```
for (ptr=arr; ptr <= &arr[SIZE-1]; ptr++)
    printf("%2c", *ptr);
```

- הדפסה ע"י קידום מצביע מכתובת התא הראשון עד כתובת התא האחרון

מערכים מצביעים ופונקציות - הסבר

- כפי שהוסבר בעבר כאשר מעבירים מערך לפונקציה, השינויים שנעשה בפונקציה ישפיעו על המערך המקורי.
- זה נובע מכך שלפונקציה מועברת הכתובת בזיכרון של המערך המקורי, והיא פועלת ישירות על ערכיו (ולא על העתק שלהם).
- כפי שאמרנו, הפונקציה הבאה תאפס מערך שמועבר אליה:

```
void zero_arr (int arr[])
{
    int i;
    for(i=0 ; i<SIZE; i++)
        arr[i]=0;        //*(arr+i)=0
}
```

מערכים מצביעים ופונקציות - הסבר

- אמרנו בעבר שכשמעבירים מערך לפונקציה שינויים שנעשה בפונקציה ישפיעו על המערך המקורי.
- זה נובע מכך שלפונקציה מועברת הכתובת בזיכרון של המערך המקורי, והיא פועלת ישירות על ערכיו (ולא על העתק שלהם).
- כפי שאמרנו, הפונקציה הבאה תאפס מערך מועבר:

```
void zero_arr (int *arr)
{
    int i;
    for(i=0 ; i<SIZE; i++)
        *(arr+i)=0;     //arr[i]=0;
}
```

נציין שאם פונקציה מצפה לקבל מצביע, אפשר להעביר אליה מערך מהסוג הזה, כי בשני המקרים מה שמועבר הוא כתובת בזיכרון.

מערכים מצביעים ופונקציות - דוגמה


```
#include <stdio.h>
#define SIZE 10
void fill_arr1(int arr[])
 {
     int i;
     for(i=0 ; i<SIZE; i++)
          *(arr+i)=i;
}
void fill_arr2(int *arr)
{
     int i;
     for(i=0 ; i<SIZE; i++)
          *(arr+i)=i*2;
}
```

```
void main()
 {
   int i,arr[SIZE];
   fill_arr1(arr);
   for(i=0;i<SIZE;i++)
        printf("%4d",*(arr+i));
   fill_arr2(arr);
   printf("\n\n");
   for(i=0;i<SIZE;i++)
        printf("%4d",arr[i]);
   printf("\n\n");
   return 0;
}
```

מצביעים ופונקציות – החלפת ערכים ללא תא עזר - דוגמה


```
#include <stdio.h>

void swap(int *ptr1, int *ptr2)
{
   *ptr1 = *ptr1 + *ptr2;
   *ptr2 = *ptr1 - *ptr2;
   *ptr1 = *ptr1 - *ptr2;
}
```

```
void main()
 {
   int num1=5,num2=10;
   swap(&num1,&num2);
   printf("num1=%d,num2=%d\n"
  ,num1, num2);
   return 0;
 }
```

מערכים ופונקציות - הסבר

- באחריותנו להפעיל אותן אך ורק על מחרוזת, כלומר על רצף של תווים שמסתיים ב- '\0', ולא סתם על מצביע לתו בודד, כי אז נקבל תוצאות שגויות אם אין את תו הסיום.
- כמו-כן, כזכור, אם פונקציה מוגדרת על מערך בגודל מסוים, ניתן להעביר אליה גם מערך בגודל אחר של אותו טיפוס, גדול יותר או קטן יותר, (כי כאמור מה שמועבר זה רק כתובת ההתחלה).
- שוב, לתשומת ליבכם !!! באחריותנו בלבד שתתבצע בפונקציה פעולה הגיונית ושהיא, הפונקציה, תדע מה גודל המערך ולא תחרוג ממנו לעולם.

מערכים ופונקציות – נקודה לתשומת לב

- אם ננסה להחזיר מפונקציה מערך שהוגדר בתוכה או מצביע על מערך כזה, אז נקבל שגיאה.

```
int * zero_array()
{
    int array[100]={0};
    return array;
}
```

- זה מכיוון שמוחזרת כתובת של המערך, אבל כל מה שהוגדר בתוך הפונקציה נמחק כשהיא מסתיימת, לכן לא תוחזר כתובת המערך.

מצביעים ומערכים - סיכום

- משתנה מערך הוא סוג של מצביע: הוא מכיל כתובת (של התא הראשון במערך), אבל אי-אפשר לשנות אותה.
- זו הסיבה שבהעברת מערך לפונקציה המערך המקורי מושפע מהשינויים שנעשים עליו.
- אפשר לגשת לתא במערך גם בעזרת חישוב כתובתו יחסית לכתובת התחלת המערך.
- למשל:

```
*(arr+5)=100;
*(arr+3)=80;
```

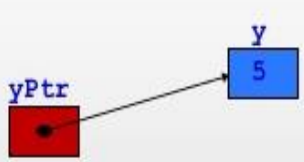
סיכום ביניים

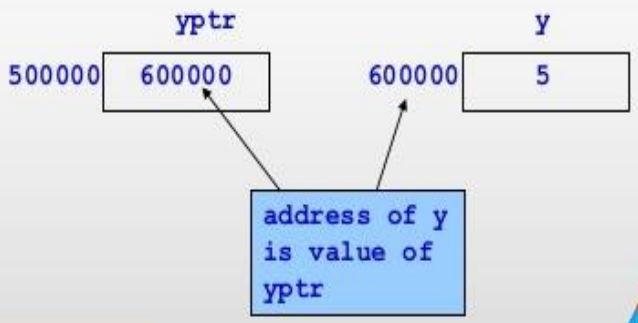
Pointer Operators

- & (address operator)
  - Returns memory address of its operand
  - Example

```
int y = 5;
int *yPtr;
yPtr = &y;
```

- yPtr "points to" y
- * - indirection/ dereferencing operator)
- *yPtr returns y dereferenced pointer is lvalue
  - *yptr = 9 ??





- כתובות בזיכרון
- שימוש במצביעים
- מערכים ומצביעים
- מערכים ופונקציות

קלט מחרוזת כולל רווחים

במידה ורוצים לבצע קלט של מחרוזת הכולל בתוכו רווחים

באמצעות הפונקציה scanf(); יש להשתמש בפונקציה:

```
fgets(str, SIZE, stdin);
```

המשמעות של הפונקציה היא שהיא קולטת מהמסך stdin לתוך

המחרוזת str 11 תווים כולל סימן ה- '\n'.

המימוש של קלט המחרוזת מתבצע ע"י:

```
if (strlen(str) > 0 && str[strlen(str)-1] == '\n')
   str[strlen(str)-1] = '\0';
```

תרגיל 1 – בדיקת זהות מחרוזות

כתבו פונקציה המקבלת שתי מחרוזות מהתוכנית הראשית ובודקת האם הן זהות בתוכנן באופן לקסיקוגרפי, מבחינת תווים, אם כן, הפונקציה תחזיר 1, אם לא, אז יוחזר 0.

הערה: יש להגביל אורך כל מחרוזת ל-10 תווים באופן הבא:

```
char str[SIZE];
scanf("%10s",str);
```

ניתן ואף מומלץ לקלוט את המחרוזת ע"י הפונקציה gets(str).

פתרון תרגיל 1 – באמצעות מערך


```
#include <stdio.h>
#include <string.h>
#define SIZE 11
int compare (char s1[ ], char s2[ ])
{
   int   i;
   for ( i=0; s1[i] != '\0' || s2[i] !=  '\0'; i++ )
         if ( s1[i] != s2[i] )
               return 0;
   return 1;
}
```

פתרון תרגיל 1 – באמצעות מערך


```
void main()
{
   char  str1[SIZE], str2[SIZE];
   printf("Enter two strings:\n");
   gets(str1);
   gets(str2);
   if(compare(str1,str2))
         printf("The two strings are the same!!!\n");
  else
         printf("The two strings are not the same!!!\n");
}
```

פתרון תרגיל 1 – באמצעות מצביעים


```
#include <stdio.h>
#include <string.h>
#define SIZE 11
int compare (char *s1,char *s2)
{
   int   i;
   for ( i=0; *(s1+i) != '\0' || *(s2+i) !=  '\0'; i++ )
         if ( *(s1+i) != *(s2+i) )
               return 0;
   return 1;
}
```

פתרון תרגיל 1 – באמצעות מצביעים


```
void main()
{
   char  str1[SIZE],str2[SIZE];
   printf("Enter two strings:\n");
   scanf("%10s%10s",str1,str2);
   if(compare(str1,str2))
         printf("The two strings are the same!!!\n");
  else
         printf("The two strings are not the same!!!\n");
}
```

תרגיל 2 – חיפוש תת מחרוזת במחרוזת

כתבו פונקציה המקבלת שתי מחרוזות מהתוכנית הראשית ובודקת האם המחרוזת האחת, הקצרה, נמצאת כתת מחרוזת בתוך המחרוזת השנייה, הארוכה.

אם כן, הפונקציה תחזיר מצביע למקום זה.

אם לא, הפונקציה תחזיר NULL.

פתרון תרגיל 2


```
#include <stdio.h>
#define SIZE 30
char *str_str( char *str1, char *str2)
{
        char *cp = str1, *s1=NULL, *s2=NULL;
        if (*str2 == '\0')
            return str1;
        while ( *cp != '\0' )
        {
                s1 = cp;
                s2 = str2;
                while ( *s1 != '\0' && *s2 != '\0' && *s1 == *s2 )
                        { s1++; s2++; }
                if ( *s2 == '\0' )
                        return cp;
                cp++;
        }
        return NULL;
}
```

בדיקת זהות תווים בדיקת המחרוזת עד לסיומה

בדיקת תת המחרוזת

פתרון תרגיל 2


```
void main()
{
   char string1[SIZE]=“You are a girl and I am a boy";
   char string2[SIZE ]="and";
   printf("%s\n",str_str(string1,string2));
}
```

תרגיל 3

כתבו פונקציה המקבלת כקלט מחרוזת ובודקת האם היא פלינדרום.

הגדרת מחרוזת פלינדרום: מחרוזת שניתן לקרוא אותה מימין ומשמאל ולקבל את אותה מחרוזת כלומר סדר האותיות ישר והפוך הינו זהה.

לדוגמה: ABCDCBA, WERFREW

הערה:

יש להגביל את אורך המחרוזת ל-100 תווים בלבד בקלט.

פתרון תרגיל 3


```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define SIZE 100
int palindrome(char *s1)
{
        int len=0;
        char *s2=NULL;
        while(*s1)
        {
                s1++;
                len++;
        }
```

בדיקת אורך המחרוזת

פתרון תרגיל 3


```
s2=s1-1;
s1=s1-len;
while(*s1)
{
        if (*(s1++) != *(s2--))
        {
                printf("The string is not a palindrome!\n");
                return 0;
        }
}
printf("The string is a palindrome!\n");
}
```

הצבת המצביע בכתובת התו האחרון במחרוזת

הצבת המצביע בכתובת התא הראשון במחרוזת

בדיקת חוסר זהות בין התווים תוך כדי קידום מצביע מהתחלת המחרוזת וחזרה של מצביע מסוף המחרוזת

פתרון תרגיל 3


```
int main()
{
        char str[SIZE];
        printf("Enter string up to 100 chars\n");
        scanf("%100s",str);        //gets(str);
        palindrome(str);
        return 0;
}
```

תרגיל 4

כתבו פונקציה המקבלת מחרוזת של אותיות ומחשבת כמה מילים נמצאות בתוך המחרוזת הנתונה.

רמז:

המילים במחרוזת מופרדות על ידי רווחים, כך שבין כל שתי מילים קיים רווח אחד בלבד !!!.

לא לשכוח כי המילה האחרונה מסתיימת עם הסימן '\0'.

ניתן לבדוק האם התו אינו שווה ל- רווח ' ' או לחילופין ניתן להשתמש בפונקציה הבודקת האם קיים רווח בתא:

isspace(ch)

שנמצאת בספריה:

```
#include <ctype.h>
```

פתרון לתרגיל 4


```
int word_cnt2(char str[])
{
   int    cnt = 0, i=0;
   if (str[i] == '\0')        // s[0] points to empty string
       return 0;
   while(str[i])          // until string ends
   {
         if (str[i] == ‘ ‘)
               cnt++;
         i++;
   }
   return ++cnt;
}
```

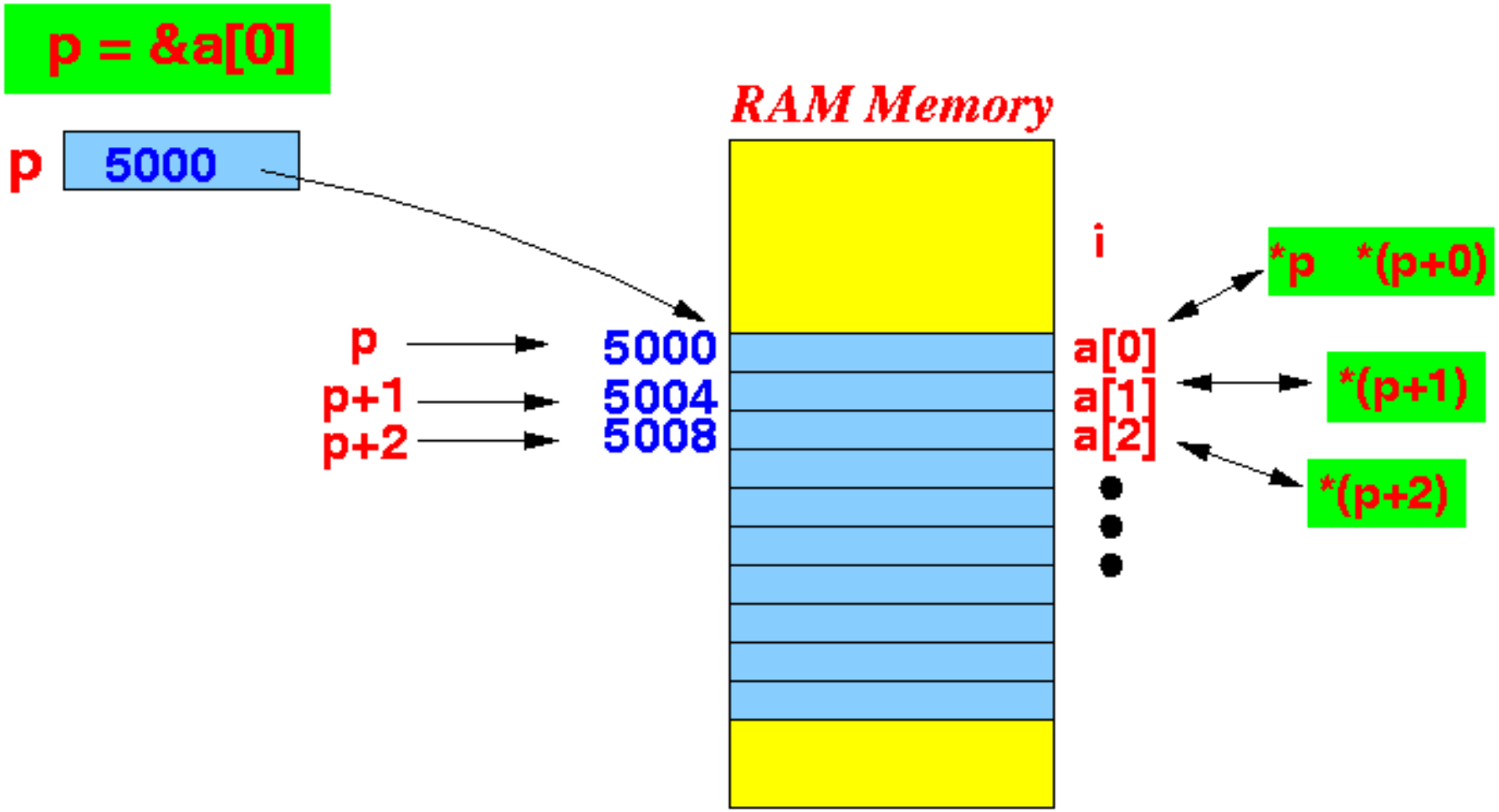
פתרון נוסף לתרגיל 4


```
int word_cnt1(char *str)
{
   int    cnt = 0;
   char  *nextchar = str + 1;
   if (*str == '\0')                    // *s points to empty string
       return 0;
   while(*nemist != '\0')               // until string ends
   {
          if (!isspace(*str) && isspace(*nextchar))
               cnt++;
          str++;  nextchar++;
   }
   if (!isspace(*str))
          cnt++;
   return cnt;
}
```

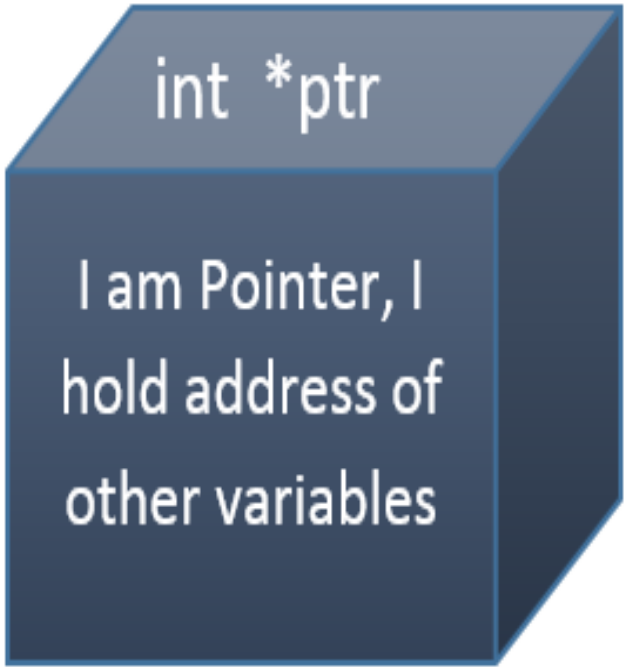
פתרון נוסף לתרגיל 4


```
int word_cnt2(char *str)
{
   int    cnt = 0;
   if (*str == '\0')        // *s points to empty string
       return 0;
   while(*str)                    // until string ends
   {
         if (*str == ‘ ‘)
               cnt++;
         str++;
   }
   return ++cnt;
}
```

מצביעים - סיכום



מצביעים - סיכום



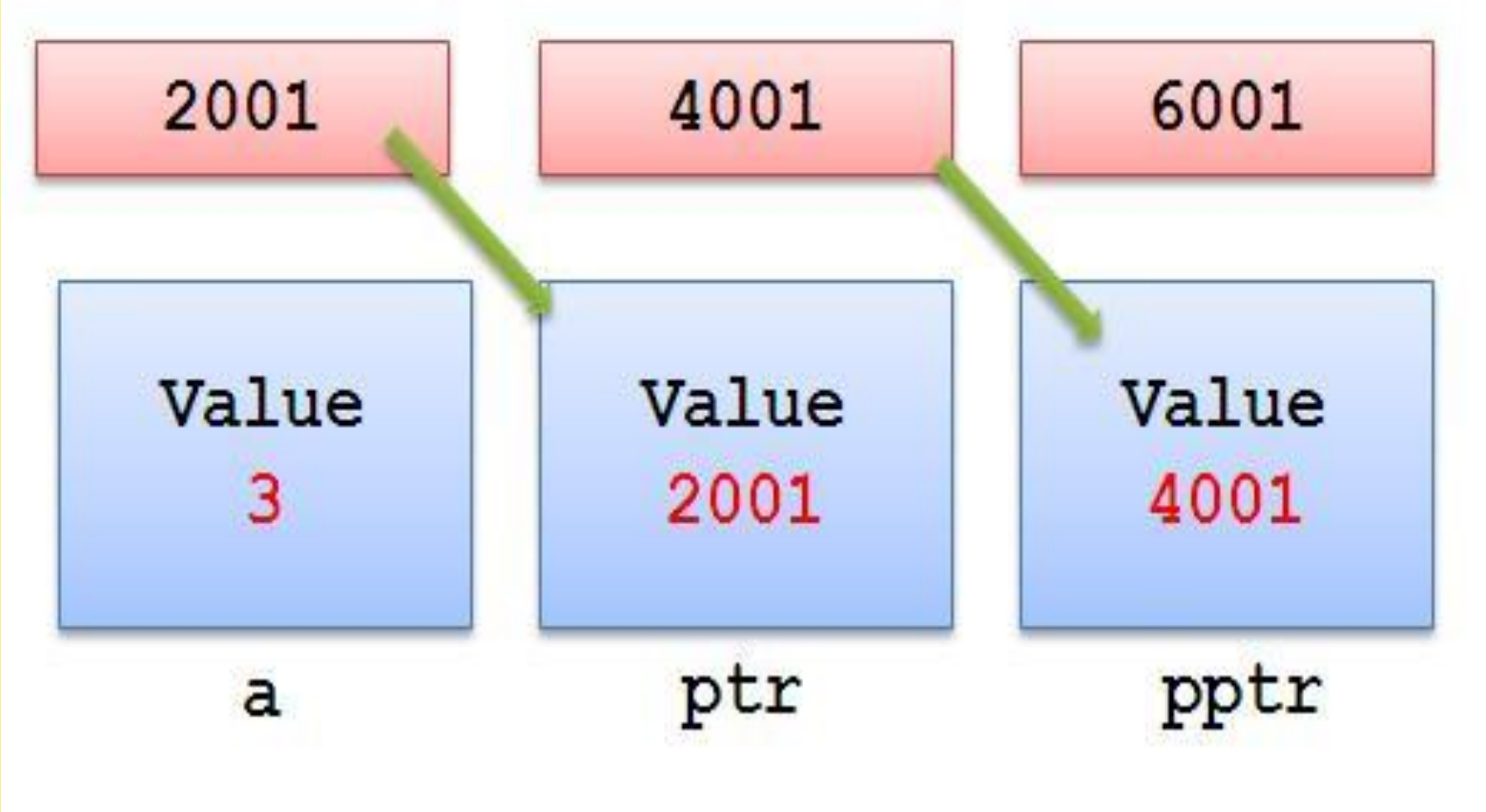
- מהן כתובות בזיכרון
- טיפוס משתנה – גודל בזיכרון
- פעולות עם כתובות
- שמירת משתנה בכתובת מסוימת בזיכרון
- טבלת כתובות המשתנים וגודלם
- מהם מצביעים ומה אפשר לעשות איתם
- מצביעים ופונקציות
- מצביעים ומערכים ומחרוזות

שאלות?

מצביעים למצביעים

- במצביעים יש אפשרות לבצע הצבעה פנימית של מצביע למצביע.
- זה ניתן לביצוע ע"י הגדרת מצביע למצביע ובכך להגדיל את רמת העקיפות בתכנית.
- כך ניתן להגדיר מערך דו ממדי ולטפל בו ע"י תנועה של מצביע למצביע באופן רציף.
- הדבר מאפשר הגדרת מערך דו ממדי ע"י הגדרת מצביע רגיל, שיציין את השורות ובתוכו מצביע פנימי המציין את תחילת כל שורה פנימית.
- כך ניתן לשלוח מערך דו ממדי לפונקציה ולטפל בפונקציה במערך ע"י שימוש במצביע למצביע.

מצביעים למצביעים - פונקציות



מצביעים למצביעים

- מגדירים מצביע למצביע בצורה דומה להגדרת מצביע רגיל, לדוגמא:

```
int  **pptr=NULL;
```

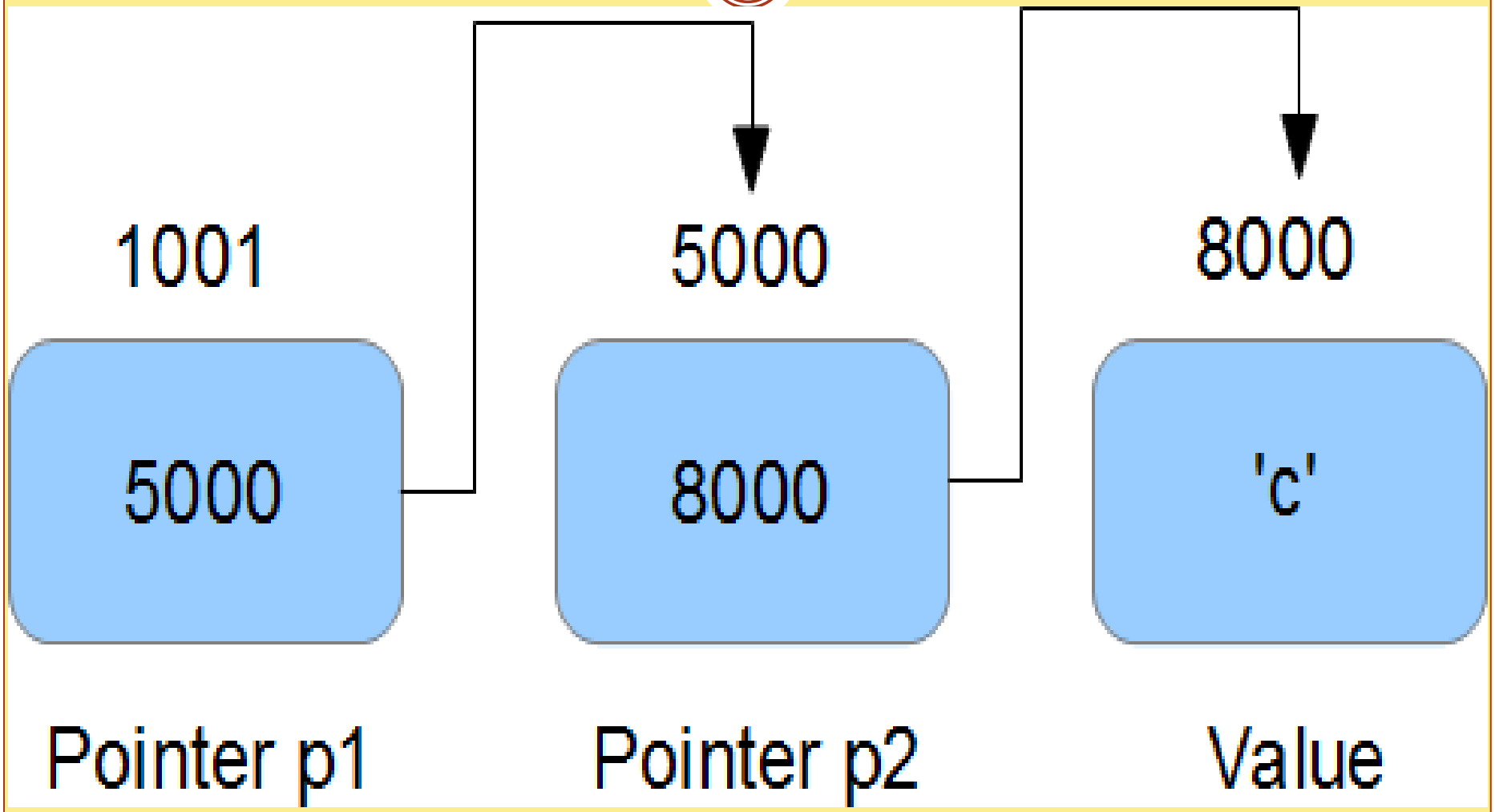
- באופן זה ניתן לקבל את כתובתו של משתנה מסוג מצביע, לדוגמא:

```
int  i;
int *pi = &i;
int  ** ppi = &pi;
**ppi  = 4;          /* i = 4 */
```

מצביעים למצביעים

- מצביע, כמו כל משתנה, צריך להיות מאוחסן במקום כלשהו בזיכרון, כיוון שהוא צריך לאחסן מידע (במקרה זה כתובת).
- אפשר לבדוק היכן המצביע עצמו מאוחסן – בעזרת האופרטור & כמובן.
- איזה טיפוס יחזיר האופרטור & הפעם? ובכן לפי הכלל, אם המצביע שלנו הוא מטיפוס * int , הרי שהטיפוס שהאופרטור & יחזיר הוא i**. (כלומר מצביע למצביע).

מצביעים למצביעים



מצביעים למצביעים - כתובות


```
#include <stdio.h>
#include <conio.h>
void main()
{
        int mis=2015;
        int *ptr1 = &mis;
        int **ptr2 = &ptr1;
        //Print address
        printf("Address of mis: %p\n", &mis);
        printf("Address of ptr1: %p\n", ptr1);
        printf("Address of ptr1: %p\n", &ptr1);
        printf("Address of ptr2: %p\n", ptr2);
        printf("Address of ptr2: %p\n", &ptr2);
}
```

Addresses:

Address of mis: 0032FA04

Address of ptr1: 0032FA04

Address of ptr1: 0032F9F8

Address of ptr2: 0032F9F8

Address of ptr2: 0032F9EC

מצביעים למצביעים - ערכים


```
#include <stdio.h>
#include <conio.h>
void main()
{
        int mis=2015;
        int *ptr1 = &mis;
        int **ptr2 = &ptr1;
        //Print values
        printf("Value of mis: %d\n", mis);
        printf("Value of ptr1: %d\n", *ptr1);
        printf("Value of ptr2: %d\n", **ptr2);
}
```

Values:

Value of mis: 2015

Value of *ptr1: 2015

Value of **ptr2: 2015

מצביעים למצביעים

- ניתן להגדיר מחוון שיצביע על הכתובת של מחוון אחר.
- לדוגמא **p היא הגדרה של מחוון למחוון.
- דוגמא לשימוש במחוון למחוון:

```
int mis,*pmis=NULL,*ppmis=NULL;
    // המחוון קיבל את הכתובת של המשתנה
pmis=&mis;
המחוון למחוון קיבל את הכתובת של
ppmis=&pmis; //המחוון
*ppmis=10;    // mis = 10
```

מצביעים למצביעים - מחוון


```
#include <stdio.h>
#include <conio.h>
void main()
{
   int mis,*pmis=NULL,*ppmis=NULL;
   pmis=&mis;            //  המחוון  קיבל את הכתובת של המשתנה
   ppmis=&(*pmis);   //המחוון למחוון קיבל את הכתובת של המחוון
   *ppmis=10;
//Print address
   printf("Address of mis: %p\n", &mis);
   printf("Address of *pmis: %p\n", pmis);
   printf("Address of *pmis: %p\n", &pmis);
   printf("Address of *ppmis: %p\n", ppmis);
   printf("Address of *ppmis: %p\n", &ppmis);
}
```

Addresses:

Address of mis: 003CFE80
Address of *pmis: 003CFE80
Address of pmis: 003CFE74
Address of *ppmis: 003CFE80
Address of ppmis: 003CFE68

מצביעים למצביעים - ערכים


```
#include <stdio.h>
#include <conio.h>
void main()
{
 int mis,*pmis=NULL,*ppmis=NULL;
 pmis=&mis;        //  המחוון  קיבל את הכתובת של המשתנה
 ppmis=&(*pmis);   //המחוון למחוון קיבל את הכתובת של המחוון
 *ppmis=1980;
  //Print Values
  printf("Value of mis: %d\n", mis);
  printf("Value of *pmis: %d\n", *pmis);
  printf("Value of *ppmis: %d\n", *ppmis);
}
```

Addresses:

Value of mis: 1980

Value of *pmis: 1980

Value of *ppmis: 1980

מצביעים למצביעים

- ניתן גם להגדיר מחוון שיצביע על מערך של מחוונים.
- מחוון למחוון רושמים ** .

```
int **ptr=NULL,*arr[4];
ptr=arr;
```

- בשורה הראשונה הגדרנו שני משתנים מטיפוס שלם.
- האחד הוא מצביע למצביע בשם ptr ולאחריו הגדרנו מערך של מצביעים בגודל 4 בשם arr .
- בשורה השנייה העברנו ל ptr את הכתובת של המצביע הראשון במערך המצביעים שהוא שם המערך arr.

מצביעים למצביעים – דוגמה עם מערך


```
#include <stdio.h>
#define ROW 2
#define COL 3
void main()
{
        int numbers[ROW][COL] = {{23,12,37},{45,54,62}}, i,j;
        numbers[1][0] = 99;
        *(*(numbers+1)+2) = 88;
        for(i=0;i<ROW;i++)
        {
            for(j=0;j<COL;j++)
                printf("%2d ", numbers[i][j]);    //*(*(numbers+i)+j)
                printf("\n");
        }
}
```

```
23 12 37
99 54 88
```

מצביעים למצביעים – דוגמה נוספת


```
#include<stdio.h>
void main()
{
    char **ptr = NULL;
    char *p = NULL;
    char ch = 'd';
    p = &ch;
    ptr = &p;
    printf("\n ch = [%c]\n",ch);
    printf("\n *p = [%c]\n",*p);
    printf("\n **ptr = [%c]\n",**ptr);
}
```

```
ch = [d]
*p = [d]
**ptr = [d]
```

מצביעים למצביעים - פונקציות


```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
void twoPoint(char **a)
{     *a += 2; }
int main()
{
    char *b[SIZE] = {"15913.178"};
    int tmp;
    tmp = atoi(*b);
    twoPoint(b);
    printf("Send [%d] to: twoPoint() => %s\n",tmp, *b);
    return 0;
}
```

שאלות?

מצביעים למצביעים - סיכום

- במצביעים יש אפשרות לבצע הצבעה פנימית של מצביע למצביע ע"י הגדרת מצביע למצביע ובכך להגדיל את רמת העקיפות בתכנית.
- הדבר מאפשר הגדרת מערך דו ממדי ע"י הגדרת מצביע רגיל, שיציין את השורות ובתוכו מצביע פנימי המציין את תחילת כל שורה פנימית.
- מצביע, כמו כל משתנה, צריך להיות מאוחסן במקום כלשהו בזיכרון, כיוון שהוא צריך לאחסן מידע (במקרה זה כתובת).
- אפשר לבדוק היכן המצביע עצמו מאוחסן – בעזרת האופרטור &.

תרגיל כיתה

1. כתוב תוכנית הקולטת מחרוזת של תווים באורך של לא יותר מ- 15 תווים מהמשתמש. התוכנית מעבירה את המחרוזת לפונקציה שהופכת אותה, מהסוף להתחלה, כל זאת ללא שימוש במחרוזת עזר נוספת וללא שימוש ב- strrev( ).

יש להשתמש בפונקציה לצורך ההפיכה ובמצביעים לצורך התנועה על המחרוזת.

2. כתוב תוכנית שתפקידה למחוק רווחים ממחרוזת קלט. התוכנית תקלוט מחרוזת ותפעיל פונקציה אשר תבדוק לאחר קליטת כל תו האם הוא רווח, במידה וכן לא תשבץ אותו במערך במידה והוא שונה מרווח תשבצו במערך. ניתן להניח כי אורך המחרוזת הוא 15 תווים בלבד ויש להשתמש בפונקציה לצורך הבדיקה ובמצביעים לצורך התנועה על המחרוזת.

תרגיל כיתה

3. כתוב תוכנית ובה פונקציה בשם sumDiff, המקבלת שני מצביעים לשלמים. לאחר הקריאה לפונקציה והפעלתה, יצביע המשתנה הראשון על תוצאת הכפל של שני השלמים, והשני על המנה של חלוקת הראשון בשני.

יש להשתמש במצביעים בתוך הפונקציה ולדאוג כי התשובות יוחזרו למשתנים המקוריים.

4. כתוב תוכנית ובה פונקציה בשם pointSort, המקבלת שלושה ערכים שלמים למשתנים וממיינת אותם מהקטן לגדול ע"י שימוש במצביעים בלבד.

יש להשתמש במצביעים בתוך הפונקציה ולדאוג כי התשובות יודפסו מקטן לגדול.

תרגיל כיתה

5. כתוב תוכנית הקולטת מחרוזת בת 20 תווים ומילה נוספת בת שתי אותיות ושולחת את שתי המחרוזות לפונקציה. הפונקציה תחפש במחרוזת הגדולה את המילה ע"י שימוש במצביעים. אם המילה נמצאה יש להחזיר את הכתובת שלה, אחרת יש להחזיר NULL.

6. כתוב תוכנית המגדירה ומפעילה פונקציה בשם: strlen_without_ch, המקבלת מחרוזת תווים, עוברת ובודקת את תווים המחרוזת, הפונקציה תספור ותדפיס את מספר התווים במחרוזת שאינם האות 'A' או 'a'. יש להשתמש בפונקציה לצורך הבדיקה ובמצביעים לצורך התנועה על המחרוזת.

תרגיל כיתה

7. כתוב תוכנית המגדירה ומפעילה פונקציה בשם: strcpy_without_ch, המעתיקה מחרוזת אחת לשנייה, העתקה מתבצעת תו אחרי תו כל עוד התווים המועתקים אינם 'M' או 'm'. בסיום תדפיס הפונקציה את המחרוזת החדשה שהתקבלה. יש להשתמש בפונקציה לצורך הבדיקה ובמצביעים לצורך התנועה על המחרוזת.

8. כתוב תוכנית המפעילה פונקציה הקולטת מערך חד ממדי באורך 10 הכולל מספרים שלמים בלבד, ממיינת אותו מקטן לגדול ע"י שימוש בפונקציה ומצביעים בלבד. התוכנית תציג את המערך לפני המיון ולאחריו.

תרגיל כיתה

9. כתוב תוכנית המפעילה פונקציה הקולטת מערך דו ממדי בגודל 5x5 הכולל מספרים שלמים בלבד וממיינת אותו מקטן לגדול ע"י שימוש בפונקציה ומצביעים בלבד. יש לדאוג כי התוכנית תציג את המערך לפני המיון ולאחריו.

10. כתוב תוכנית המפעילה פונקציה המקבלת מערך דו ממדי שנקלט בתוכנית הראשית והמכיל 10 מחרוזות של מילים בנות 7 אותיות כל אחת.

הפונקציה ממיינת את המערך לקסיקוגרפית בסדר עולה ע"י החלפת כתובות בלבד של מצביעים. יש לדאוג כי התוכנית תציג את המערך לפני המיון ולאחריו.

תרגיל כיתה

11. כתוב תוכנית המפעילה פונקציה המקבלת שתי מחרוזות שנקלטו בתוכנית הראשית ובודקת האם הן אנגרמה ומציגה הודעה מתאימה. אנגרמה פרושו ששתי המחרוזות מכילות את אותן אותיות אך בסדר שונה.

לדוגמה המחרוזת: creative ו- reactive הם אנגרמה.

12. כתוב תוכנית המפעילה פונקציה המקבלת מחרוזת שנקלטה בתוכנית הראשית ובודקת ע"י בניית היסטוגרמה של תווים כמה פעמים מופיע כל תו במחרוזת ומדפיסה טבלה לאחר סיום הבדיקה.

תרגיל כיתה

13. כתוב תוכנית המפעילה פונקציה המקבלת מחרוזת של אותיות ומבצעת עיבוד תמלילים כך שלאחר העיבוד יורדו מהמחרוזת כל העיצורים והיא תוצג כמחרוזת חדשה ללא העיצורים שהיו במקורית.

לדוגמה:

Enter a string to delete vowels

How will an English sentence without vowels?

String after deleting vowels:

Hw wll n nglsh sntnc wtht vwls?

תרגיל כיתה

14. כתוב תוכנית המפעילה פונקציה המקבלת מספר ויוצרת משולש פסקל באמצעות שימוש במצביעים בלבד. התוכנית קולטת מהמשתמש מספר שלם, המספר נשלח לפונקציה והיא מייצרת משולש פסקל בגובה המספר שנקלט מהמשתמש כמצביע.

15. כתוב תוכנית הקולטת שני מערכים של תווים וממיינת אותם מהקטן לגדול. התוכנית תפעיל פונקציה המקבלת שלושה מערכים חד ממדים, שניים שנקלטו מהמשתמש ואחד חדש ותמזג אותם כשהם ממוינים למערך שלישי חדש.

הערה: כיוון ששני המערכים שנקלטו ממוינים בסדר עולה, לכן גם המערך החדש הממוזג יהיה מערך ממוין בסדר עולה.

שאלות?