# פרק 11
# מחרוזות



## מחרוזות (Strings)

- אנו עובדים עם טקסט במחשב כל הזמן – אנו מדפיסים טקסט למסך, קוראים אותו מהמקלדת או מקובץ, ומעבדים אותו למטרות שונות.
- אנו רוצים אפשרות לאחסן ולעבוד עם קטעי טקסט גדולים בצורה נוחה.
- **מחרוזת** היא טיפוס המיועד לייצוג טקסט חופשי, כלומר רצפי תווים.
- פעולות אופייניות על מחרוזות כוללות קריאת מחרוזת מהקלט, כתיבת מחרוזת לפלט, שכפול מחרוזת, השוואה בין מחרוזות, גישה לתו כלשהו במחרוזת ועוד.



## מחרוזות ב-C

- בשפת C אין טיפוס ייעודי לייצוג מחרוזות.
- מחרוזת מיוצגת פשוט על ידי רצף תווים בזיכרון (מהטיפוס **`char`**) שהתו האחרון שבהם הוא תו מסיים מיוחד הנקרא **`null`**.
- התו **`null`** הינו תו מיוחד שערכו המספרי הוא 0 (בטבלת ASCII ובכל טבלת קידוד בכלל). תו זה תפקידו לציין את סופן של מחרוזות בזיכרון, וניתן לסמנו ב-C כמספר 0, או עם גרשיים כך: **`'\0'`**.
- כל העבודה עם מחרוזות ב-C נעשית דרך מצביעים מטיפוס **`char*`**, שמצביעים לתחילת המחרוזת בזיכרון (לתו הראשון בה).
- <u>חשוב לזכור</u>: לא כל רצף תווים בזיכרון הוא מחרוזת! על מנת שרצף תווים יוכל לשמש כמחרוזת, הוא חייב להסתיים בתו **`null`**.



## מחרוזות ב-C

- הנה תמונת הזיכרון עבור מחרוזת לדוגמה. מחרוזת זו מייצגת את המילה **`"yoink!"`**. המחרוזת מורכבת מ-7 תווים בסך הכול (מטיפוס **`char`**) ולכן תופשת 7 בתים בזיכרון:

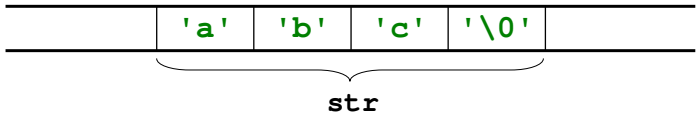

- נניח עתה ש-**`ptr`** הוא מצביע מטיפוס **`char*`** לתו הראשון במחרוזת הזו. במקרה זה **`ptr`** נחשב גם **כמצביע למחרוזת**.
- המחרוזת ש-**`ptr`** מצביע עליה בזיכרון הינה מהתו **`'y'`** שמוצבע ע"י **`ptr`**, ועד ל-**`0`** הראשון שאחריו בזיכרון.

# מחרוזות ב-C

- כיצד נוכל ליצור מחרוזות בעצמנו? הדרך הנפוצה ביותר היא באמצעות מערכים של **char**. נתבונן בהגדרה הבאה:

```
char str[] = { 'a', 'b', 'c', '\0' };
```

- פקודה זו מקצה זיכרון לרצף של 4 תווים שהאחרון שבהם הוא **null**, ולכן היא יוצרת מחרוזת. שם המערך (**str**) משמש גם כמצביע לתחילת המחרוזת. תמונת הזיכרון היא:




- כיוון שהתו **null** ערכו המספרי הוא 0, הגדרה שקולה הינה:

```
char str[] = { 'a', 'b', 'c', 0 };
```



# מחרוזת לעומת מערך

נדגיש את האבחנה בין "מחרוזת" לבין "מערך של **char**":

- **מחרוזת** היא כתובת של רצף תווים בזיכרון, שהאחרון שבהם הוא 0. היא עשויה להיות מאוחסנת במערך, אבל ייתכן באותה מידה שהיא הוקצתה באופן אחר. יתר על כן, גם אם היא מאוחסנת במערך, לא כל המערך נחשב בתור המחרוזת, אלא רק קטע המערך עד התו 0. מערך שאין בו את התו 0 איננו נחשב כמחרוזת כלל.
- למשל, נתבונן בהגדרה הבאה:

```
char smiley[10] = { ':' , ')' , 0 };
```

- אורך המערך **smiley** הוא 10.
- אורך המחרוזת שמאוחסנת ב-**smiley** הוא 2 שכן המחרוזת כוללת שני תווים בלבד (האפס המסיים איננו נספר).



# אתחול מערך למחרוזת

- ב- C ישנה דרך נוחה לאתחל מערך של **char** למחרוזת כלשהי. לשם כך, פשוט נרשום בשורת האתחול את תווי המחרוזת בין צמד מירכאות כפולות " " :

```
char smiley[10] = ":)";
```

- פקודה זו גורמת לתוכנית להעתיק את התווים **':'** ו- **')'** לתוך המערך **smiley**, וכן להוסיף לאחריהם את התו המסיים 0 באופן אוטומטי. למעשה, אתחול זה **שקול לחלוטין** לזה שראינו בשקף הקודם. אילו היינו כותבים כך:

```
char smiley[] = ":)";
```

היה מוקצה למערך **smiley** בדיוק גודל הזיכרון הדרוש לאחסון המחרוזת (כולל התו המסיים 0), דהיינו 3 בתים במקרה זה.



# אתחול מערך למחרוזת

- שימו לב שאופן הכתיבה שראינו בשקף הקודם, שנראה כמו "השמה" של מחרוזת לתוך מערך, אפשרי רק בשלב האתחול של המערך. לא ניתן לבצע בצורה דומה השמה של מחרוזות בזמן ריצת התוכנית.
- למשל, הקוד הבא לא יעבוד (בין היתר, כיוון שמערך הוא כזכור קבוע ולא ניתן לבצע השמה לתוכו) :

```
smiley = ":(";
```

- על מנת להעתיק מחרוזת אחת לתוך מחרוזת אחרת, יש לכתוב פונקציה ייעודית שמבצעת העתקה זו (בהמשך..).

# גישה לאברי המחרוזת

- כיוון שמחרוזת היא פשוט מצביע מטיפוס **char***, אנו כבר יודעים כיצד לגשת לאברי המחרוזת: ניתן לעשות זאת באמצעות האופרטור **[]**, או לחילופין להשתמש בחשבון מצביעים .
- למשל, הקוד הבא מדפיס את המחרוזת שבמערך **smiley**, תוך ניצול העובדה שהמחרוזת מסתיימת באפס:

```
int i=0;
while (smiley[i]) {
  putchar(smiley[i++]);
}
```

- את פנים הלולאה היינו יכולים לכתוב גם בעזרת חשבון מצביעים:

```
putchar(*(smiley + i++));
```



# גישה לאברי המחרוזת

- נאתחל עתה מצביע לתחילת המחרוזת, כך:

```
char *ptr = smiley;
```

- פעולה זו יוצרת לנו דרך נוספת לגשת למחרוזת, באמצעות המצביע **ptr**. אנו יכולים, למשל, לממש את הדפסת **smiley** באמצעות מצביע שסורק את המחרוזת:

```
char *ptr = smiley;
while (*ptr) {
  putchar(*(ptr++));
}
```



# קבועים ב-C

- **קבוע** הוא כל פריט מידע שבזמן הקומפילציה "נתפר" ישירות לתוך קוד המכונה של התוכנית. בניגוד למידע רגיל, שנשמר בתוך משתנה בזיכרון, זהו מידע קבוע, שנמצא בתוך קוד התוכנית ממש.
- ניתן לזהות קבועים על ידי התבוננות בקוד: כל ערך שמופיע בו באופן מפורש הוא קבוע ש"נצרב" בקוד המקומפל. למשל, בקוד הבא, הקבוע המספרי 2 איננו נשמר בזיכרון בתוך משתנה כלשהו, אלא הוא חלק מן הקוד המקומפל של התוכנית:

```
tmp = 2*tmp;
```

- ראינו גם קבועי תווים בקוד (זכרו שהטיפוס שלהם למעשה **int**!). למשל, בחישוב ערכה המספרי של ספרה על פי קוד ה-ASCII שלה:

```
digit = character - '0';
```



# קבועי מחרוזת

- כאשר עבדנו עם קבועי תווים, במקום לכתוב בכל פעם את קוד ה-ASCII של התו, פשוט כתבנו את התו עצמו בתוך צמד גרשיים **' '** והקומפיילר החליף אותו אוטומטית בערך המספרי המתאים.
- שפת C מאפשרת לנו ליצור גם קבועי מחרוזת בקוד בצורה נוחה.
- על מנת לציין קבוע מחרוזת, נכתוב בקוד את המחרוזת הרצויה, ונשים סביבה צמד מירכאות כפולות **""**.
- כל קבוע מחרוזת כזה שמופיע בקוד, יהפוך למערך של **char** ש"תפור" בתוך קוד המכונה של התוכנית. מערך זה מכיל בדיוק את כל התווים שבמחרוזת, ואת התו 0 המסיים.
- שימו לב שוב להבדל בין הקבועים **'a'** ו-**"a"**! הראשון מייצג תו בודד (מספר ASCII), והטיפוס שלו הוא **int**. השני הוא מערך של **char**, והוא מכיל שני איברים: **{'a',0}**.

# קבועי מחרוזת

איפה משתמשים בקבועי מחרוזת?

- בפקודות כמו **printf()** ו-**scanf()** אנו מוצאים מחרוזות בקרה קבועות (זהו השימוש הנפוץ ביותר):

```
printf("I have %d camels", 100);
```

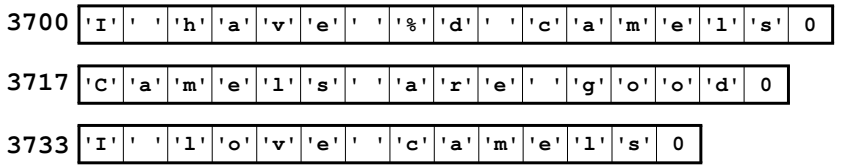
- ניתן גם לאתחל מערכים, ומצביעים ל-**char**, לקבועי מחרוזת:

```
char sarr[] = "Camels are good";
char * sptr = "I love camels";
```



# מנגנון הפעולה של קבועי מחרוזת

1. עם תחילת ריצת התוכנית, כל קבועי המחרוזת בתוכנית נכתבים לאזור קבועים מיוחד בזיכרון:




2. בזמן ריצת התוכנית, כל מחרוזת קבועה בקוד מוחלפת אוטומטית במצביע מטיפוס **char***, המציין את מיקומה של המחרוזת בזיכרון הקבועים.



# מנגנון הפעולה של קבועי מחרוזת

עבור המקרה שלנו, ההחלפות ייראו כך:

- **printf()** תקבל מצביע למיקום מחרוזת הבקרה שלה בזיכרון:

```
printf( (char*)3700, 100);
```

- **sptr** יאותחל עם הכתובת של המחרוזת המתאימה בזיכרון הקבועים:

```
char * sptr = (char*)3717;
```

- רק במקרה של אתחול מערך, מנגנון הפעולה הוא שונה. שימו לב שבמקרה זה **המערך עצמו** (שנמצא במחסנית ולא בזיכרון הקבועים!) צריך להכיל את מחרוזת האתחול שציינו. לכן, בזמן ריצת התוכנית, כאשר המערך מוקצה, מתבצעת פעולת העתקה של תוכן מחרוזת האתחול, מזיכרון הקבועים אל תוך המערך.

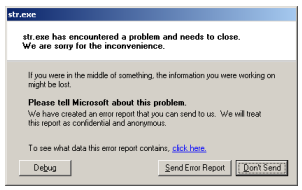


# מנגנון הפעולה של קבועי מחרוזת

- חשוב לדעת שהזיכרון בו נשמרות המחרוזות הקבועות הוא בדרך כלל זיכרון המיועד לקריאה בלבד, וכל ניסיון לכתוב אליו יגרור שגיאת זמן ריצה.
- למשל, בקוד הבא עלולה להיגרם שגיאת זמן ריצה, כיוון שאנו מנסים דרך המצביע **sptr** לכתוב לזיכרון מוגן:

```
char * sptr = "I love camels";
sptr[0] = 'U'
```

- ב-Dev-Cpp, הרצת הקוד הנ"ל גורמת לעצירת התוכנית ולהקפצת החלון הבא כאשר התוכנית מגיעה לשורת ההשמה:



# מנגנון הפעולה של קבועי מחרוזת

- שימו לב שהקוד הבא, לעומת זאת, הוא בטוח, ולא יגרור שגיאה:

```
char sarr[] = "Camels are good";
sarr[0] = 'c'
```

- הסיבה היא שבמקרה זה אנו כותבים לתוך המערך **sarr**, שהוא מערך רגיל שמוקצה במחסנית של התוכנית, ולכן אין כאן שגיאה. למעשה, הקשר היחיד בדוגמה זו לזיכרון המוגן הוא שעם הקצאת **sarr**, המערך קיבל עותק של מחרוזת האתחול, שהיא אכן מאוחסנת בזיכרון הקבועים.



# העברת מחרוזת לפונקציה

- כפי שניתן בוודאי לנחש, מעבירים מחרוזת לפונקציה פשוט על ידי העברת הכתובת שלה (=הכתובת של התו הראשון בה).
- למשל, נכתוב פונקציה שסופרת את מספר התווים במחרוזת. הפונקציה תסרוק את המחרוזת עד שתגיע ל-0 המסיים, ותספור:

```
int strlen(char *s)
{
  int i=0;
  while (s[i]) {
    i++;
  }
  return i;
}
```

הלולאה נמשכת כל עוד **s[i]** איננו התו 0 המסיים את המחרוזת



# העברת מחרוזת לפונקציה

- דוגמאות לשימוש ב-**strlen()**:

```
char s[50] = "hello there";

printf("s:   %d\n", strlen(s));
printf("s+6: %d\n", strlen(s + 6));
```

- הפלט יהיה:

```
s:   11
s+6: 5
```

- שימו לב ש-**s+6** הוא למעשה מצביע למחרוזת **"there"**, שאורכה 5.



# הדפסת מחרוזת

- הפרמטר הראשון של **printf()** הוא מטיפוס מחרוזת – כלומר מטיפוס **char***. עד כה שמנו תמיד מחרוזת קבועה בתור פרמטר זה, אולם ניתן לשים כל מחרוזת אחרת במקומה. לפיכך, ניתן להדפיס מחרוזת פשוט ע"י העברתה כפרמטר הראשון של **printf()**, כך:

```
char s[50] = "hello there";

printf(s);
putchar('\n');
printf(s+6);
```

- הפלט יהיה:

```
hello there
there
```

# הדפסת מחרוזת

- הבעיה בטכניקה הקודמת היא ש-**printf()** אינה מדפיסה את מחרוזת הבקרה שלה כמו שהיא, אלא היא מזהה תגי בקרה במחרוזת, ומדפיסה אותם בהתאם לפרמטרים נוספים שמועברים לפונקציה.
- למשל, שימו לב לקוד הבא:

```
char password[50] = "r*34x%dGh";
printf(password);
```

- הרצת קוד זה ב-DevCpp הניבה את הפלט:

```
r*34x2089878893Gh
```

- הסיבה היא שבמחרוזת **password** הופיע הצמד **%d**, אולם לא ציינו כל מספר שלם כפרמטר נוסף ל-**printf()**! לכן תג הבקרה הוחלף בתוכן אקראי, ובמקרה זה במספר 2089878893.



# הדפסת מחרוזת

- דרך טובה יותר להדפיס מחרוזת היא באמצעות תג הבקרה **%s**, המציין מחרוזת בפונקציות קלט/פלט. במקרה זה המחרוזת המועברת ל-**printf()** מודפסת כמו שהיא, ולא נוצרת הבעיה שראינו. למשל:

```
char password[50] = "r*34x%dGh";

printf("pass   = %s\n", password);
printf("pass+6 = %s\n", password + 6);
```

- נקבל את הפלט:

```
pass   = r*34x%dGh
pass+6 = dGh
```



# קריאת מחרוזת

- ניתן גם לקרוא מחרוזת מהקלט, באמצעות התג **%s**.
- התג **%s** מדלג על כל הרווחים בתחילת הקלט, ואז קורא את כל התווים שנמצאים בקלט עד לרווח הבא (כלומר, למעשה הוא קורא מילה אחת). מילה זו, בתוספת תו 0 מסיים, נכתבת למחרוזת היעד.
- שימו לב שאין צורך לכתוב תו **&** במקרה זה, כיוון ששם המחרוזת בעצמו הוא מצביע לזיכרון שאליו יש לכתוב את המחרוזת.

```
char name[50];
printf("enter your name: ");
scanf("%s", name);
printf("name = %s\n", name);
```

```
enter your name:    shimon  gever
name = shimon
```



# קריאת מחרוזת

- נציין שבפועל, קריאת מחרוזת באמצעות התג **%s** הינה פעולה מוגבלת מאוד. בעיה רצינית במיוחד הינה הבעיה של **דריסת זיכרון**:
- כאשר **scanf()** קוראת מחרוזת, היא קוראת אוטומטית את כל המילה, וכותבת אותה לזיכרון. אולם, ברור שהזיכרון אליו המילה נכתבת מוגבל. למשל, בדוגמה הקודמת הקצנו מערך באורך 50 לקלט, מה שמאפשר לנו לקרוא לכל היותר מילה באורך 49 (זכרו שצריך מקום ל-**null** המסיים).
- אם המשתמש יקליד שם שאורכו גדול מ-49, מה שיקרה הוא ש-**scanf()** תכתוב את יתר המילה מעבר לגבולות המערך שהקצנו, ונקבל חריגת זיכרון! אין כל דרך למנוע מ-**scanf()** לעשות זאת, ולפיכך הבעיה הזו חמורה מאוד (וירוסים במחשב לעיתים מנצלים באגים שכאלו לצורך פעולתם).
- לכן, בקוד "אמיתי" אין משתמשים באופן קריאה זה של מחרוזות, ויש פונקציות מוצלחות יותר. עם זאת, אנו לא נעסוק בפונקציות אלו בקורס.

# הספרייה string.h

- כפי שניתן לראות, הכלים ששפת C מציעה לטיפול במחרוזות הם בסיסיים ביותר. לכן, השפה מגדירה קובץ ספרייה בשם **<string.h>** הכולל מספר רב של פונקציות לטיפול במחרוזות.
- הפונקציות בספרייה זו מספקות כלים שונים לטיפול במחרוזות, ומאפשרות עבודה נוחה יותר איתן.
- לדוגמה, הספרייה מכילה פונקציה בשם **strlen()** שמחזירה את מספר התווים במחרוזת כלשהי. המימוש שלה דומה מאוד לזה שראינו.
- פונקציה נוספת, שנראה עתה, משרשרת שתי מחרוזות למחרוזת אחת:

```
char* strcat(char *s1, char *s2);
```



# הפונקציה strcat()

- קיצור של *string concatenate*.
- הפונקציה מקבלת שתי מחרוזות **s1** ו-**s2**, ומעתיקה את תוכן המחרוזת **s2** בסוף המחרוזת **s1**. הפונקציה מניחה שב-**s1** יש מספיק מקום להכיל את כל התווים של **s2**.
- **strcat()** מחזירה מצביע ל-**s1**, שמכיל את תוצאת השרשור.
- דוגמה לפעולת **strcat()**:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **s1** לפני: | 'I' | ' ' | 'l' | 'o' | 'v' | 'e' | 0 | · | · | · | · | · |
| **s2**: | ' ' | 'y' | 'o' | 'u' | 0 | · | · | · | · | · | · | · |
| **s1** אחרי: | 'I' | ' ' | 'l' | 'o' | 'v' | 'e' | ' ' | 'y' | 'o' | 'u' | 0 | · |



# מימוש הפונקציה strcat()

- נתחיל במציאת התו 0 המסיים של **s1**.
- לאחר מכן, נעתיק את תוכן **s2** לתוך **s1**. נתחיל ב-0 המסיים של **s1**, שאותו יש לדרוס, ומשם נעתיק את כל התווים שבמחרוזת **s2**, כולל ה-0 המסיים שלה (שיהפוך להיות ה-0 המסיים החדש של **s1**).

```
char* strcat(char *s1, char *s2)
{
  char *pos = s1;
  while (*pos)
    pos++;
  while (*pos = *s2) {
    pos++; s2++;
  }
  return s1;
}
```

מציאת סוף **s1**

העתקת **s2**



# פונקציות נוספות ב-string.h

- הספרייה **string.h** מכילה פונקציות רבות. כדאי לבדוק את תוכן הספרייה ולנסות לממש חלק מהפונקציות שבה. נזכיר מספר פונקציות מספרייה זו שעליכם להכיר.
- פונקציות שמימשנו בפרק זה:

```
int strlen(char* str);
```

- פונקציה זו סופרת את מספר התווים (לא כולל ה-**null**) במחרוזת.

```
char* strcat(char* dest, char* src)
```

- פונקציה זו משרשרת את תוכן המחרוזת **src** לסוף המחרוזת **dest**. המימוש שלה דומה מאוד לזה של **strcat()** שראינו בשקף הקודם.

# פונקציות נוספות ב-string.h

- פונקציות נוספות:

```
char* strcpy(char* dest, char* src)
```

- פונקציה זו מעתיקה את תוכן המחרוזת **src** לתוך המחרוזת **dest**, תוך דריסת התוכן הנוכחי של **dest**. המימוש שלה דומה למדי לזה שראינו עבור **strcat()** קודם.

```
int strcmp(char* s1, char* s2)
```

- פונקציה זו משווה שתי מחרוזות באופן לקסיקוגרפי (לפי סדר מילוני, על פי טבלת ASCII). היא מחזירה 0 אם שתי המחרוזות זהות, מספר שלילי אם **s1** "קטנה יותר" (=קודמת במילון), ומספר חיובי אם **s1** "גדולה יותר" (=מאוחרת יותר במילון).



# דוגמאות ל-strcmp()

| | |
|---|---|
| `strcmp("lion", "zebra")` | `< 0` |
| `strcmp("tiger", "tiger")` | `0` |
| `strcmp("kuala", "Kuala")` | `> 0` |
| `strcmp("rat", "rat snake")` | `< 0` |
| `strcmp("jaguar?", "jaguar!")` | `> 0` |



# השוואת מחרוזות

- שימו לב שהשוואה בין מחרוזות ניתן לבצע אך ורק על ידי השוואה איבר-איבר (כפי שנעשה בפונקציה **strcmp()**, למשל).
- לשם הדוגמה, נתבונן בהגדרות הבאות:

```
char* p1 = "google";
char* p2 = "google";
char a[] = {'g','o','o','g','l','e','\0'};
```

- כעת, שימו לב לארבע ההתניות הבאות (הסבירו את התוצאה!):

```
p1 == p2
```
תלוי קומפיילר

```
p1 == "google"
```
תלוי קומפיילר

```
p1 == a
```
false

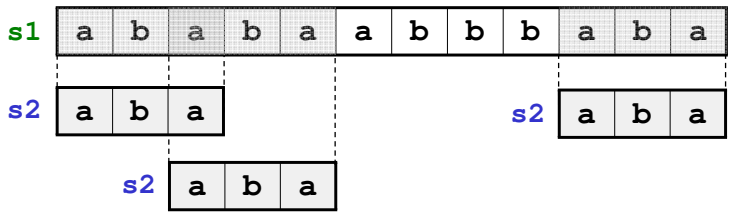
```
strcmp(p1, a)
```
0 (true דהיינו)



# דוגמה אחרונה: חיפוש תת-מחרוזת

- נכתוב פונקציה שמקבלת שתי מחרוזות **s1** ו-**s2**, ומחפשת את כל המקומות שבהם **s2** מופיעה בשלמותה ב-**s1**.
  הפונקציה תחזיר את מספר הפעמים שהיא מצאה התאמה כזו.
- למשל, בדוגמה הבאה **s2** מופיעה ב-**s1** שלוש פעמים:

```
s1 = "ababaabbbaba"
s2 = "aba"
```



# חיפוש תת-מחרוזת: שלב א'

- נתחיל מפונקציה שבודקת אם **s2** היא **רישא** של מחרוזת **s1**. כדי לבדוק זאת, נסרוק במקביל את שתי המחרוזות, ואם נמצא תו כלשהו של **s2** ששונה מהתו המתאים לו ב-**s1**, נעצור. אחרת, נמשיך בהשוואות עד שאחת המחרוזות מסתיימת: אם **s2** הסתיימה, אזי היא באמת רישא של **s1**. אחרת, **s2** ארוכה מ-**s1** ולכן אינה רישא שלה.

```
int begins_with(char *s1, char *s2)
{
  while (*s1 && *s2) {
    if (*s1 != *s2)
      return 0;
    s1++; s2++;
  }
  return (*s2==0);
}
```



# חיפוש תת-מחרוזת: שלב ב'

- נעזר בפונקציה **begins_with()** לשם כתיבת הפונקציה שלנו (זו שסופרת את מספר ההופעות של מחרוזת אחת במחרוזת שנייה):

```
int count_occurances(char *s1, char *s2)
{
  int counter  = 0,
      curr_pos = 0,
      last_pos = strlen(s1) - strlen(s2);

  while (curr_pos <= last_pos) {
    counter += begins_with(s1+curr_pos, s2);
    curr_pos++;
  }
  return counter;
}
```



# מערך של מחרוזות

- מערך של מחרוזות הוא בעצם מערך של **מצביעים** מטיפוס **char***, שכל אחד מהם מצביע למחרוזת אחרת. שימו לב שזהו מערך חד-ממדי רגיל, ולא מערך דו ממדי.
- שימו לב שכל מצביע יכול להצביע למחרוזת באורך אחר, או אפילו ל-**null**.
- דוגמה:


```
char s[] = "I'm a normal string";
char* ptrs[3];

ptrs[0] = "I am a happy pointer";
ptrs[1] = s;
ptrs[2] = 0;
```



# אתחול מערך של מחרוזות

- ניתן לאתחל מערך של מצביעים למחרוזות באמצעות רשימת איתחול, כמו כל מערך. שימו לב שהכוכבית שהוצמדה קודם לשם הטיפוס **char** מוצמדת כאן לשם המשתנה; שתי צורות הכתיבה שקולות לחלוטין:

```
char *beatles[4] = { "John Lennon",
                     "Paul McCartney",
                     "George Harrison",
                     "Ringo Starr" } ;
```

- בקוד זה, כל אחת מארבע המחרוזות בתוך ה-**{}** הינה מחרוזת קבועה. בתחילת ריצת התוכנית, כל אחת מהן נכתבת לזיכרון המוגן של התוכנית, וכאשר מערך המצביעים מוקצה, כל מצביע במערך מאותחל לכתובת בה נמצאת המחרוזת המתאימה בזיכרון הקבועים.

# עבודה עם מערך של מחרוזות

- קיבלנו מערך **beatles[]** שאיבריו הם מצביעים מטיפוס **char***.
- כל איבר **beatles[i]** במערך זה הוא מצביע, המכוון למחרוזת כלשהי בזיכרון הקבועים של התוכנית.
- ניתן לעבוד עם מצביעים אלו כמו עם מחרוזות לכל דבר. למשל:

```
for (i=0; i<4; ++i) {
  printf("beatle %d = %s\n", i, beatles[i]);
}
```

- כמו כן מותר לשנות את הכתובת שמאוחסנת בכל אחד מן המצביעים, על מנת שיצביע למחרוזת אחרת.



# עבודה עם מערך של מחרוזות

- שימו לב שכתיבת השם **beatles** ללא **[]** מחזירה כרגיל מצביע לתחילת המערך. הפונקציה הבאה מדפיסה את פרטי חברי הלהקה:

```
for (i=0; i<4; ++i)
{
  printf("Please welcome %s, who plays the %s!\n",
         *(beatles+i), (i==3)? "drums":"guitar");

  printf("Here is his name in reverse: ");

  for (j=strlen(beatles[i])-1; j>=0; j--) {
    putchar( (*(beatles+i))[j] );
  }
  printf("\n\n");
}
```



# עבודה עם מערך של מחרוזות

- גרסאות נוספות ללולאה האחרונה (הסבירו!):

```
for (j=strlen(beatles[i])-1; j>=0; j--)
  putchar(*(beatles[i] + j));
```

```
for (j=strlen(beatles[i])-1; j>=0; j--)
  putchar( *( *(beatles+i) + j) );
```

```
for (j=strlen(beatles[i])-1; j>=0; j--)
  putchar(beatles[i][j]);
```



# עבודה עם מערך של מחרוזות

- הפלט של התוכנית נראה כך:

```
Please welcome John Lennon, who plays the guitar!
Here is his name in reverse: nonneL nhoJ

Please welcome Paul McCartney, who plays the guitar!
Here is his name in reverse: yentraCcM luaP

Please welcome George Harrison, who plays the guitar!
Here is his name in reverse: nosirraH egroeG

Please welcome Ringo Starr, who plays the drums!
Here is his name in reverse: rratS ogniR
```

# העברת פרמטרים ל- main()

- ניתן לכתוב את הפונקציה **main()** כך שתוכל לקבל פרמטרים מהמשתמש בזמן שהוא מעלה את התוכנית. כיצד זה מתבצע?
- כאשר אנו מריצים את התוכנית מתוך DOS, אנו כותבים את שם התוכנית, ולאחר מכן ניתן להוסיף מספר פרמטרים כרצוננו, מופרדים על ידי רווחים. מערכת ההפעלה מעבירה פרמטרים אלו כמו שהם (כמחרוזות) לפונקציה **main()** של התוכנית.
- לדוגמה, חבילת Dev-Cpp כולל בין היתר תוכנת DOS שנקראת **gcc**. תוכנה זו מקבלת שם של קובץ C בשורת הפקודה, והיא מקמפלת אותו ויוצרת קובץ הרצה. הקריאה ל-**gcc** משורת ה-DOS נראית כך:

```
C:\Dev-Cpp\bin> gcc calc.c
```

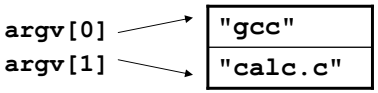


# העברת פרמטרים ל- main()

- על מנת לקבל בתוך הפונקציה **main()** את הפרמטרים שנשלחו לתוכנית, יש להוסיף בחתימת הפונקציה **main()** שני פרמטרים:

```
int main(int argc, char *argv[]) {…}
```

- ***argv*** הוא מערך של **char***, דהיינו **מערך של מחרוזות**. פרמטר זה מכיל את רשימת כל המחרוזות שנכתבו בשורת הפקודה, כולל שם התוכנית עצמה.
- ***argc*** הוא **מספר המחרוזות** שיש ב-**argv**, כלומר אורך המערך.
- למשל, בדוגמה של **gcc** שראינו קודם, **argc==2**. המחרוזות ב-**argv[]** יהיו:






# דוגמה פשוטה

- התוכנית **test** מקבלת קלט משורת הפקודה, ומדפיסה אותו :

```
int main(int argc, char *argv[])
{
  int i;
  for (i=0; i<argc; ++i)
    printf("argv[%d]: %s\n", i, argv[i]);
  return 0;
}
```

- דוגמה להרצה של **test** :

```
C:\> test one to three !
argv[0]: test
argv[1]: one
argv[2]: two
argv[3]: three
argv[4]: !
```



# דוגמה מעניינת יותר

- לשם מה אנו מקבלים את שם התוכנית עצמה ב-**argv[]** ? הנה שימוש. נניח כתבנו תוכנית המקבלת שני פרמטרים משורת הפקודה (לא כולל שם התוכנית עצמה). במקרה שמספר הפרמטרים שגוי, נדפיס הודעה ונצא:

```
int main(int argc, char *argv[])
{
  if (argc != 3)
    printf("Usage: %s <infile> <outfile>\n",
           argv[0]);
  return 0;
}
```

- דוגמה להרצה:

```
C:\> testprog one two three four
Usage: testprog <infile> <outfile>
C:\>
```

# עוד דוגמה: שוב תוכנית החיפושיות

- נשנה את תוכנית החיפושיות, כך שהיא תדפיס רק את הפרטים של חברי הלהקה שהמשתמש ציין את מספרם בשורת הפקודה (כל חבר להקה מצוין על ידי מספר בין 0 ל-3, לפי המיקום שלו במערך):

```
int main(int argc, char *argv[])
{
  int i, num;
  char *beatles[4] = {…};

  for (i=1; i<argc; ++i) {
    int num = atoi(argv[i]);
    if (num>=0 && num<=3)
      printf("You have chosen %s!\n",
             beatles[num]);
    else printf("No such beatle!\n");
  }
  return 0;
}
```

מקבלת **atoi()** מחרוזת המכילה מספר שלם, ומחזירה את הערך המספרי שלו. מוגדרת ב-**<stdlib.h>**



# עוד דוגמה: שוב תוכנית החיפושיות

- דוגמה להרצת התוכנית:

```
C:\> beatles 2 0 2 3 6 1

You have chosen George Harrison!
You have chosen John Lennon!
You have chosen George Harrison!
You have chosen Ringo Starr!
Not a beatle!
You have chosen Paul McCartney!
```