

חזרה על קלט פלט ודברים נוספים ב-C

- העברת פרמטרים לתוכניות ב-C
- קלט ופלט ב-C
- I/O redirection
- זיכרון דינמי
- מצביעים
- מערכים
- גישות לא חוקיות לזיכרון

העברת פרמטרים לתוכניות ב-C

- כאשר **main** נקראת מועברים אליה שני פרמטרים:
 - **argc** - מספר הארגומנטים בשורת הפקודה (כולל הפקודה עצמה)
 - **argv** - מצביע לערך של מחרוזות אשר מכילות את הארגומנטים בשורת הפקודה
- נכתב תוכנית בשם **echo** אשר תדפיס את הארגומנטים בשורת הפקודה שלה בשורה אחת עם רווח ביניהם
- לדוגמה:

> echo hello, world

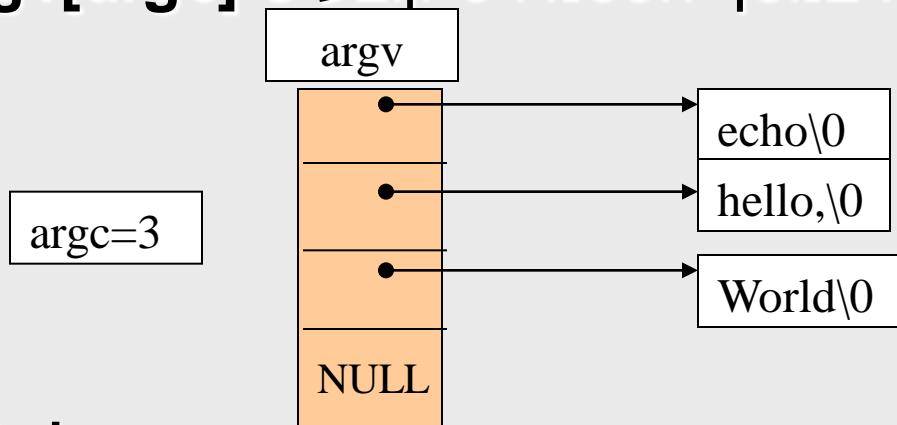
hello, world

>

➤ הוחלט ש **argv[0]** הוא שם התוכנית הנקראת. אך **argc** הוא לפחות 1. אם **argc** הוא 1 אין ארגומנטים אחרי שם התוכנית. בדוגמה לעיל **argc** הוא 3 ו **argv[0]**, **argv[1]**, **argv[2]** הם "hello," , "world" , "echo" בהתאם

העברה פרמטרים לתוכניות ב-C

- הארגומט האופציוני הראשון יהיה ב `argv[1]` והאחרון יהיה ב `argv[argc-1]`. בנוספ' הסטנדרט קובע ש `argv[argc]` יהיה מחרוזת ריקה.



```
#include <stdio.h>
/* echo command-line arguments */
int main (int argc, char* argv[])
{
    int i;
    for (i = 1; i < argc; i++)
        printf ("%s ", argv[i]);
    printf ("\n");
    return 0;
}
```

- התוכנית `:echo`

חרזה על קלט פלט ב-C

- ▶ פונקציות קלט/פלט נמצאות בספריה `stdio.h` , ככלומר צריך לבצע
`#include <stdio.h>`

- ▶ ניתן בעזרתו לבצע קלט מהמקלדת, פלט למסך וקלט/פלט לקבצים כדי להשתמש בקובץ יש צורר:
 1. לפתח את הקובץ לקריאה או לכתיבה
 2. לקרוא/לכתוב אל הקובץ
 3. לסגור את הקובץ
- ▶ כל הפעולות על קבצים נעשות ע"י משתנה מטיפוס מצבייע ל-FILE (הטיפוס FILE מוגדר ב `stdio.h`)
- ▶ לדוגמה:

```
FILE* fd; /* file descriptor */
```

פתיחת קובץ

פתיחת קובץ נעשית ע"י הפקה fopen:

FILE* fopen(const char* filename,const char* mode);

filename - שם הקובץ לפתיחת

mode - המצב בו יש לפתח את הקובץ:

"r" - פתח לקריאה בלבד

"w" - פתח לכתיבה בלבד

"a" - פתח לכתיבה בסוף הקובץ

הפקציה מחזירה מצביע לעורץ הפתוח אם הצליחה לפותחו

אחרת מחזירה NULL

לדוגמה:

FILE* fd;

fd = fopen("hello.c" , "r");

אם פותחים קובץ לכתיבה והקובץ לא קיים עדין, אז הוא נוצר בעקבות הקריאה לפקציה fopen. פתיחה לקריאה של קובץ שאינו קיים גורמת לשגיאה

סגירה של קובץ

- סגירת קובץ שנפתח ע"י fopen נעשית ע"י הfonk' `:fclose`
- **int fclose (FILE* fd);**
- מחזירה 0 אם הצליחה או EOF אם קرتה שגיאה.
- לדוגמה:

```
FILE* fd;  
fd = fopen("hello.c" , "r");  
/* Perform input/output on file */  
if (fclose(fd) == EOF)  
    printf ("Could not close file!\n");  
else printf ("File closed successfully.\n");
```

- בסיסם ריצת תוכנית מערכת דואגת לסגור את כל הקבצים
- קיימת מוגבלה על מספר הקבצים הפתוחים בו-זמנית עבור תוכנית מסוימת

ערוצים סטנדרטיים

- ב-C יש התייחסות איחודית לכל סוג הקלט והפלט. גם הקלט מהמקלדת והפלט מהמסר נעשים ע"י ערוצים ומצבייעים לFILE
- כאשר תוכנית C מתחילה לזרע מערכת הפעלה פותחת עבורה بصورة אוטומטית שלושה ערוצים סטנדרטיים:
 1. נפתח לקריאה (בד"כ מהמקלדת)
 2. נפתח לכתיבה (בד"כ אל המסר)
 3. נפתח לכתיבה (בד"כ אל המסר)
- stdin, stdout ו stderr הם למעשה שמות משתנים כאשר כולם מティיפסו מצביעים לFILE.
- לדוגמה הפקודה:

```
fclose (stdin);
```

תגרום לכך שלא ניתן יותר לקרוא קלט מהמקלדת

הערה: אם תוכנית קוראת נתונים מהמקלדת ורוצה להקליד תו EOF
יש להקיש Ctrl-d

ערוצים סטנדרטיים

► דוגמאות: התוכנית `copy`

- במידה וקיבלה שני פרמטרים, מעתיקת את הקובץ אשר שמו הינו הפרמטר הראשון, לקובץ אשר שמו הינו הפרמטר השני
- במידה וקיבלה פרמטר יחיד, מעתיקת את הקובץ אשר שמו הינו פרמטר זה לערוץ הפלט הסטנדרטי
- במידה ולא התקבלו שמות קבצים כלל, התוכנית מעתיקת את מה שמתקיים בערוץ הקלט הסטנדרטי לערוץ הפלט הסטנדרטי

ערוצים סטנדרטיים – להדפסה!!

```
#include <stdio.h>
#define CHUNK_LEN 100
void error(char *msg) {
    fprintf(stderr, "Error: %s\n", msg);
    exit(1);
}
int main(int argc, char *argv[]) {
    FILE *in; FILE *out ;
    char buf[CHUNK_LEN];
    if (argc>3) error("Bad Number of parameters");
    if (argc>1){
        in = fopen(argv[1], "r");
        if (in==NULL) error("Can't open input file");
    } else {
        error("no agev1");
    }
    if (argc>2){
        out = fopen(argv[2], "w");
        if (out==NULL) error("Can't open output file");
    } else {
        error("no argv1");
    }
    while(fgets(buf, CHUNK_LEN, in) != NULL)
        fputs(buf, out);
    fclose(in);
    fclose(out);
    return 0;
}
```

ערוצים סטנדרטיים

char* fgets (char* line, int maxline, FILE *fp)

- קוראת שורה, כולל '\n', מקובץ fp לטור line. לכל היותר maxline-1 תווים יקראו line
- מחזירה NULL בסוף קובץ ו-b-error, אחרת מחזירה line

int fputs(char* line, FILE *fp)

- כותבת את line לקובץ
- מחזירה EOF אם יש שגיאה, אפס אחרת

char* gets(char* s)

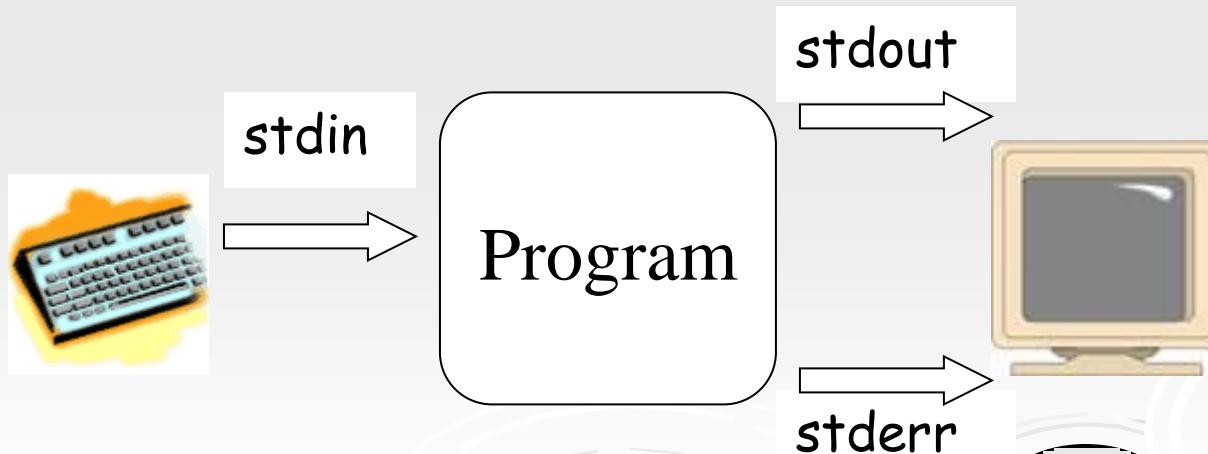
- בדומה ל-fgets רק מהקלט הסטנדרטי
- מחליף '\n' ב-'0'

int puts(char* s)

- בדומה ל-fputs רק אל הפלט הסטנדרטי
- מוסיף '\n'

I/O redirection

- ▶ בכל תוכנית שרצה תחת Unix עroz הקלט הסטנדרטי מחובר אל המקלדת, וערוצי הפלט והודעות השגיאה הסטנדרטיים מחוברים אל המסר כבירת מחדל
- ▶ ב CShell ניתן ל創ן מחדש את עroz הקלט הסטנדרטי והן את ערוצי הפלט והשגיאות הסטנדרטיים אל ומאת קבצים כלשהם.
הcreation מחדש נקרא redirection



Input redirection

<program> < <filename>

- תרגום ל- program לקבל את הקלט הסטנדרטי שלו מהקובץ filename
- לדוגמה:

% copy <filename

- תרגום ל-copy הנטונה ללקחת את הקלט מהקובץ filename
- הפעלה בצורה הנ"ל שקופה (מבחינת התוצאה) עברו התוכנית copy להפעלה:

% copy filename

- אך מה ההבדל מבחינת התוכנית?

Output redirection

<program> > <filename>

תרגום ל- program לכתוב את הפלט הסטנדרטי שלו לקובץ filename. אם קיימں כבר קובץ בשם filename לפניו ביצוע הפקודה, הפקודה לא תבוצע ➤

% cat myfile

line 1

line 2

% cat myfile > out

% cat out

line 1

line 2

%

Output redirection

<program> >> <filename>

מכוננת את הפלט הסטנדרטי של program לקובץ filename או
משרשת אותו לסוף הקובץ. אם לא קיים קובץ בשם filename
לפני ביצוע הפקודה, הפקודה לא תבוצע

% cat yourfile

line 3

% cat yourfile >> out

% cat out

line 1

line 2

line 3

%

Output redirection

- ראיינו ששתי פקודות CiOן הפלט הקודמות עלולות להימשך. הראשונה (>) אם הקובץ קיים, והשנייה (>>) אם הקובץ אינו קיים
- ניתן להבטיח שהפקודות הנ"ל יצליחו בכל מקרה ע"י הוספה ! לפקודות CiOן:

<program> >! <filename>

- הורסת את הקובץ filename וכתבת עליו את הפלט הסטנדרטי של program בכל מקרה
- אם הקובץ filename אינו קיים, אז מתקבלת אותה התנהגות כשל >

<program> >>! <filename>

- יוצרת את הקובץ filename אם הוא לא קיים.
- אם הקובץ filename קיים, אז מתקבלת אותה התנהגות כשל >

Input and output redirection

- ניתן להפנות גם את ערזע הקלט הסטנדרטי של התוכנית וגם את ערזע הפלט
- לדוגמה פקודות:

% cat < in > out

% cat < in >! out

גורמת להעתיקת קובץ in לקובץ out (מה ההבדל?)

% cat < in >> out

% cat < in >>! out

גורמת להוספת קובץ in לסוף קובץ out (מה ההבדל?)

Multiple redirection

```
#include <stdio.h>
int main() {
    fprintf(stderr, "This is error\n");
    printf("This is stdout\n");
    return 0;
}
```

```
> gcc prog.c -o prog
> prog
This is error
This is stdout
> prog > out.txt
This is error
> cat out.txt
This is stdout
> prog >& errout.txt
> cat errout.txt
This is error
This is stdout
> (prog > out) >& err
> cat out
This is stdout
> cat err
This is error
```

ניהול זיכרון של תוכניות ב-C

- משתנים ב C נבדלים האחד מהשני בשני אופנים:
 - **טיפוס** – למשל `int`, `char` או `double`. שונים האחד מהשני מבחינה הערךיים האפשריים עבור כל אחד, אילו פעולות ניתן לבצע על כל אחד מהם, וכמוות הזיכרון שנדרשת עבור כל אחד מהם
 - **אופן הקצאה**
- ניתן להקצות משתנים ב 4 אופנים:
 - משתנים גלובליים
 - משתנים לוקאלים
 - משתנים סטאטיים
 - משתנים דינמיים

ניהול זיכרון של תוכניות ב-C

- **משתנים גלובליים** - משתנים אשר מוגדרים לאורך כל התוכנית וניתן לגשת אליהם מכל הfonקציות של התוכנית. המשתנים מוקצים באופן אוטומטי כאשר התוכנית מתחליה, ונשמרים לאורך כל הריצה
- **משתנים מקומיים** - משתנים פנימיים של fonקציות, רק הfonקציה שבה הם מוגדרים יכולה לגשת אליהם. המשתנים אלו מוקצים בכל פעם שהfonקציה נקראת, ומשוחרים כאשר הfonקציה מסתיימת
- **משתנים סטטיסיים** - משתנים פנימיים של fonקציות. המשתנים אינם נשמרים על ערכם בין קריאות שונות לfonקציה, ומשוחרים רק עותק אחד של כל משתנה לאורך כל התוכנית.
- מבין שלושת הקבוצות הללו סוג המשתנים השימושי ביותר הנה המשתנים המקומיים. שימוש בשאר סוגי המשתנים נחשב לרוב **כתנות רע**.

משתנים דינמיים

- ▶ הבעיה עם משתנים לוקאליים הנה כי המשתנים נהרסים כאשר פונקציה נגמרה. לעיתים נרצה כי משתנים מסוימים ישארו בזיכרון גם לאחר שהפונקציה נגמרה, יתר על כן לעיתים נרצה להקצות בזמן ריצה מספר רב של משתנים אשר מספרם אינו ידוע בזמן כתיבת התוכנית אלא רק בזמן ריצה
- ▶ **משתנים דינמיים** - מוקצים ע"י התוכנית בזמן ריצה. המתכנת מէיצה את המשתנים מאייזור בזיכרון המכונה ערימה (Heap) ובאחריות המתכנת לשחרר את המשתנים שהוקצו לאחר שאין בהם צורך
- ▶ אי שחרור זכרון שהוקצה עלול לגרום לתוכנית להכשל עקב חוסר בזיכרון
- ▶ הקצתה הזכרון מתבצעת ע"י **malloc**.
- ▶ שחרור הזכרון מתבצע ע"י **free**.

מבנה הזיכרון

- תוכנית המחשב רואה את הזיכרון כמערך גדול, של בתים (bytes)
- לכל בית יש כתובות ייחודית לו
 - התוכנית יכולה להיעזר בבית אחד או יותר בכדי לשמר משתנים

נדרש בית אחד עבור ערך
מסוג. char

נדרשים 4 בתים עבור ערך
מסוג. int

שמות
משתנים

עריך	כתובת
(זבל)	0x200
'a'	0x201
?	0x202
6	0x203
0	0x204
0	0x205
0	0x206
?	0x207
...	...

מבנה הזיכרון

- תוכנית המחשב רואה את הזיכרון כמערך גדול, של בתים (bytes)
 - לכל בית יש כתובת ייחודית לו
 - התוכנית יכולה להיעזר בבית אחד או יותר בכדי לשמר משתנים
- זכרון המחשב מחולק עבורי התוכנית לשני חלקים עיקריים
 - חלק המנוהל באופן אוטומטי ע"י התוכנית שם נשמרים המשתנים הגלובליים, הлокאלים והסתטואטיים
 - חלק (ערימה) - חלק המנוהל ע"י המתכנת במפורש
 - ניתן לבקש להקצות חלק זה שטחי זיכרון בגודלים שונים ע"י הפקודה `malloc`
 - לשחרר זיכרון אשר אין בו צורך יותר ע"י הפקודה `free`
- פעולה ההקצאה מחייבת את הכתובת של תחילת שטח הזיכרון שהוקצה. בכדי לשמר כתובת זו יש להיעזר במשתנה מסווג מצביע

מצביעים

► משתנה מסוג מצביע

משתנה שערך הוא כתובת בזיכרון שבתוכה נמצא נתון כלשהו
<type name>*<variable name>

המשתנה מיועד להצביע על מקום בזיכרון המכיל המשתנה מטפו
<type name>

int *ptr;

ptr הוא משתנה אשר מיועד להצביע על מקום בזיכרון מטפו **int**

► כתובות

נניח שהגדכנו משתנה ע"י

int x;

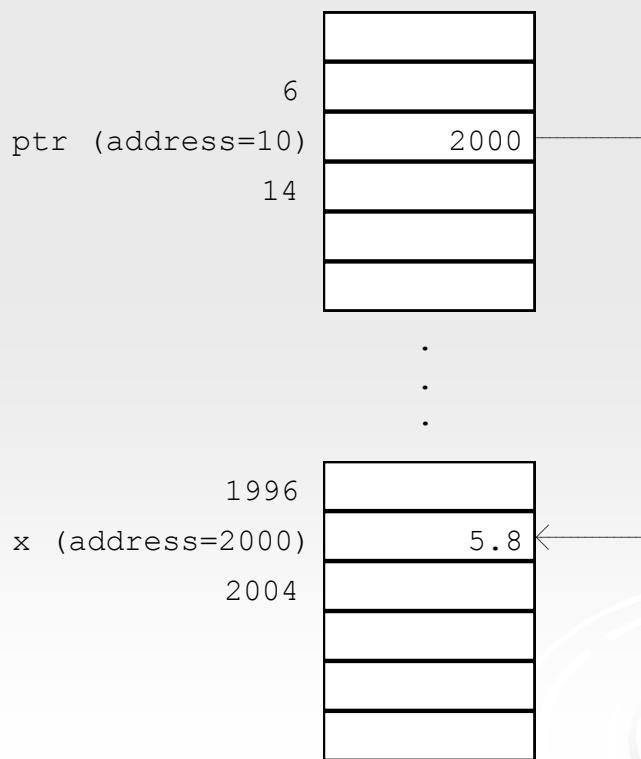
הביתוי:

&x

הוא הכתובת בזיכרון בה שמור תוכנו של **x**
למשל, אם **x** הוא שלוש ומוחסן בכתובת 2000 אז ערכו של
הביתוי **&x** הוא 2000

מצביעים

```
float x;  
float* ptr;  
ptr = &x;  
*ptr=5.8;  
.  
.
```



➤ **שימוש במשתנה מסוג מצביע**
גישה למשתנה בעזרת מצביע
לכתובתו

➤ הפקודה האחרונה שකולה ל: `x=5.8`.
➤ שימוש לב כי מצביע הוא משתנה בעצמו
ולכן גם הוא נשמר בזיכרון.

מצביים ומצביעים

- **הגדרת מצביע אינה גורמת ליצירת המקום אשר אליו הוא מצביע**
- **int *pi;**
- **וז אינו מצביע למקום בזיכרון המכיל integer אלא רק יכול להצביע**
למקומות בזיכרון המכילים int
- ***pi = 0;**
- **הצבת 0 למקום אליו מצביע וק עלולה לגרום להעפת התוכנית כיון**
שלא ידוע להיכן בזיכרון וק מצביע וגישה למקום אליו הוא "מצביע"
עלולה להיות גישה לא חוקית לזכרו
- **לפניהם גישה למקום בזיכרון המצביע ע"י מצביע יש לוודא כי המצביע**
מצביע לכטובת חוקית. יש לאותחל מצביעים כך שייכלו את הערך
NULL לפניהם שמכנסת אליהם כתובות חוקית, בצורה זאת ניתן
לבדק האם הם אינם מצביעים לכטובת חוקית

```
int j ;  
int *pi ;  
pi = & j ;  
*pi = 0;
```

מצביעים ומערכות

- יצירת מערך עם איברים כאשר ערכי האינדקסים נעים בין 0 ל number-1

<type name> <variable> [<number>]

int x[3];

- גישה למקומות במערך (חדר-ממדי) ע"י:

x[2];

- "שם המערך" הנה מצביע אשר לא ניתן לשנות את המקום אליו הוא מצביע. لكن הביטוי x ללא אינדקסים מהוות מצביע אל המקום הראשון במערך! כלומר, $x=3$ * שקול ל $x[0]=3$

- הגדרת מערך בגודל number גוררת הקצתה number אברים מהטפס המתאים, בעוד שהגדרת מצביע אינה גורמת להקצת האיבר המצביע

- ניתן להציבו למערך (או איבר במערך) גם ע"י מצביע

int *p, *q ;

p = x ; /* p points to x[0] */

q = x+2 ; /* q points to x[2] */

p = q-1; /* p points to x[1] */

מצביים והקצאות זיכרון דינמיות

`void* malloc (unsigned nbytes)`

- אם הקצאה הצליחה מוחזר מצביע לקטע בזיכרון, אחרת מוחזר הערך NULL
- כיוון שהזיכרון המוקצה יכול לשמש לשימירת ערכים מטיפוס כלשהו, הטיפוס המוחזר ע"י `malloc` הנה `* void` הנ"ו `malloc` הינו `hstdlib.h`.
- הגדרות של `malloc` ו `NULL` נמצאים ב `hstdlib.h`. לכן, כדי להשתמש בהם יש לבצע:
- לדוגמה:

```
int *ptr;  
ptr=(int*)malloc(4);           /* allocate 4 bytes */  
if (ptr==NULL)                /* check if allocated */  
    error();
```

- כתת `ptr` מצביע למקום בזיכרון שבו ארבעה בתים פנויים (אם הקצאה הצליחה)
- כיוון ש `ptr` הינו מטיפוס `int*` ו `malloc` מחרירה ערך מטיפוס `* void`, יש לבצע `casting`

מצביים והקצאות זיכרון דינמיות

- כדי למנוע את הצורך בשינוי תוכנית בגלל גודל שונה של טיפוסים, בმადრიმ შონიმ, ניתן להשתמש באופרטור `sizeof`

sizeof(<type name>)

- מוחזיר את מספר הבתים של הטיפוּם במערכת הנוכחית

```
ptr=(int*)malloc(sizeof(int));  
*ptr=3;
```

כעת ניתן להשתמש ב `ptr`* כ משתנה מסוג int (אם ההקצתה הצלילה)

מה הטעות במקרה זה?

```
ptr=(int*)malloc(sizeof(int *));
```

מצביים והקצאות זיכרון דינמיות

free(<ptr var>)

- משחרר את בлок הזיכרון שהוקצה ושם המשתנה מצביע עליו. יש לשחרר כל בлок זיכרון שהוקצה, **לאחר סיום השימוש בו**, כדי למנוע מצב של חוסר זיכרון. דוגמא:

```
ptr=(int *)malloc(sizeof(int));  
if (ptr==NULL)  
    error();  
/* use ptr */  
free (ptr);
```

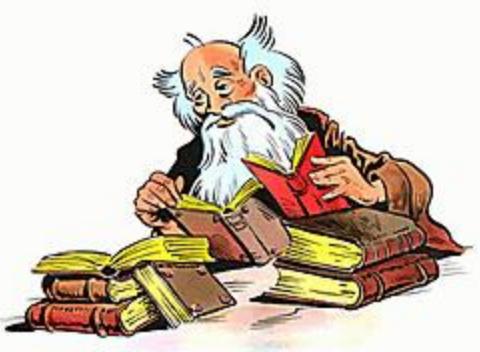
גישה לא חוקית לזכרו

טעויות נפוצות



גישה לא חוקית לזיכרון

- מצביע יכול להכיל כל כתובת שהוא
 - זיכרון שלא הוקצה לשימוש
 - זיכרון שמשמש לדברים אחרים
- התוכנית עלולה לקרוס – Bus error
- התוכנית עלולה לרוץ כרגע
 - אבל... ריצה מוצלחת על קלט אחד לא מבטיחה כלום על קלט אחר
- כל גישה (קריאה או כתיבה) דרך מצביע שאינו מצביע לכתובת חוקית עלולה לגרום לשגיאה



טעויות נפוצות

➤ הקצאת מחרוזת קצרה מדי

- לא לשכח את '\0'
- strlen איננה סופרת את '\0' כאשר היא מוחזירה אורך של מחרוזת

➤ שימוש שגוי ב-sizeof

- האופרטור מקבל טיפוס של משתנה
 - `ptr = (int*)malloc(sizeof(int *));`
 - `ptr = (int*)malloc(sizeof(int));`



טעויות נפוצות - המשך

➤ חריגה מגבולות המערך

- הקומפיילר לא שומר את האינפורמציה מהו גודל המערך ולכן אינו מתריע בזמן קומפיילציה על גישה מחוץ לגבולות המערך

➤ הזבל לסל וחסול

- אחרי free אי אפשר להשתמש בזיכרון, גם לא ממצבייע אחר



טעויות נפוצות - המשך

- שימוש במשתנים לוקליים של פונקציה מחוץ לפונקציה
 - יצירת משתנה לוקלי בפונקציה, והחזרת הכתובת שלו לפונקציה הקוראת
 - ההבדל בין `=` ל- `==`
- `if(ptr == NULL) printf("Bad pointer!\n");`
- `if(ptr = NULL) printf("Bad pointer!\n");`

דוגמה – חריגה מוגבלות המערכת

להדפסו!!

```
#include <stdio.h>
#define TITLE ("\n\tMemory Corruption: How easy it is\n\n")
#define BUF_SIZE (8)
int main()
{
    int i;
    char c[BUF_SIZE], buf[BUF_SIZE];
    printf("Type string c[]\n");
    gets(c);
    printf("%s\n",c);
    printf("Type string buf[]\n");
    gets(buf);
    printf("%s\n",buf);
    for (i=4; i > -10 ; i--){
        c[i] = 'a';
        printf("i= %2d, buf = %s c= %s\n", i, buf, c);
    }
    return 0;
}
```

Program mem_corruption.c

(output)

buf →

Memory Corruption: How easy it is

Type string c[]

012

Type string buf[]

0123456

I = 4, buf = 0123456 c = 012

I = 3, buf = 0123456 c = 012aaÿ•

I = 2, buf = 0123456 c = 01aaaÿ•

I = 1, buf = 0123456 c = 0aaaaÿ•

I = 0, buf = 0123456 c = aaaaaÿ•

I = -1, buf = 0123456aaaaaaÿ• c = aaaaay•

I = -2, buf = 012345aaaaaaaaÿ• c = aaaaay•

I = -3, buf = 01234aaaaaaaaÿ• c = aaaaay•

0
1
2
3
4
a X
a X
a X
a X
a X
a X
a X
זבל a
זבל

c →