בס"ד

עבודה מסכמת – מערכות הפעלה

שאלה 1:

```
char globBuf[65536];    /* 1. Where is allocated? */
```

1. המשתנה globBuf נמצא ב uninitialized data segment (bss).
לפי מבנה הזכרון של תכניות בשפת c, משתנים גלובליים יכולים להימצא או ב uninitialized data segment או ב initialized data segment כאשר המשתנה ממוקם בחלק הראשון אם המתכנת לא איתחל אותו ובחלק השני כאשר המתכנת איתחל את המשתנה.
מאחר ו globBuf לא אותחל אז הוא ממוקם ב uninitialized data segment.
*מופיעים בסוף הקובץ צילומי מסך שמאשרים קביעה זו.

```
int primes[] = { 2, 3, 5, 7 };  /* 2. Where is allocated? */
```

2. המשתנה primes נמצא ב initialized data segment.

לפי מבנה הזכרון של תכניות בשפת c, משתנים גלובליים יכולים להימצא או ב uninitialized data segment או ב initialized data segment כאשר המשתנה ממוקם בחלק הראשון אם המתכנת לא איתחל אותו ובחלק השני כאשר המתכנת איתחל את המשתנה.

מאחר ו primes אותחל אז הוא ממוקם ב initialized data segment.

*מופיעים בסוף הקובץ צילומי מסך שמאשרים קביעה זו.

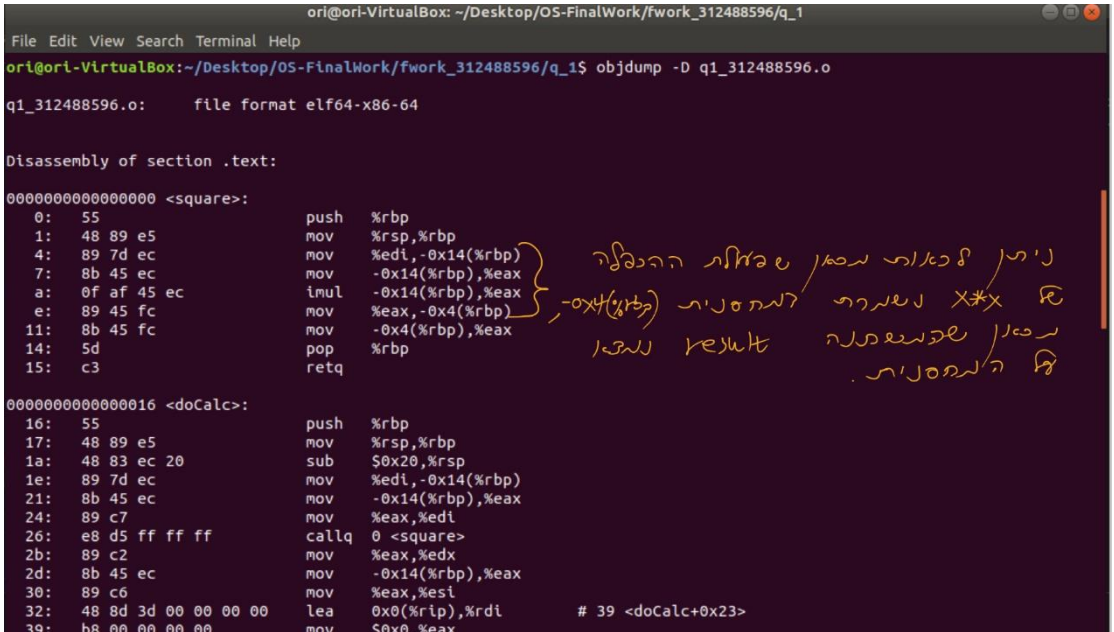
```
static int square(int x) /* 3. Where is allocated? */
```

3. הפונקציה square נמצא ב text segment.
לפי מבנה הזכרון של תכניות בשפת c, הוראות של פונקציות נמצאות ב text segment.

```
int result;             /* 4. Where is allocated? */
```

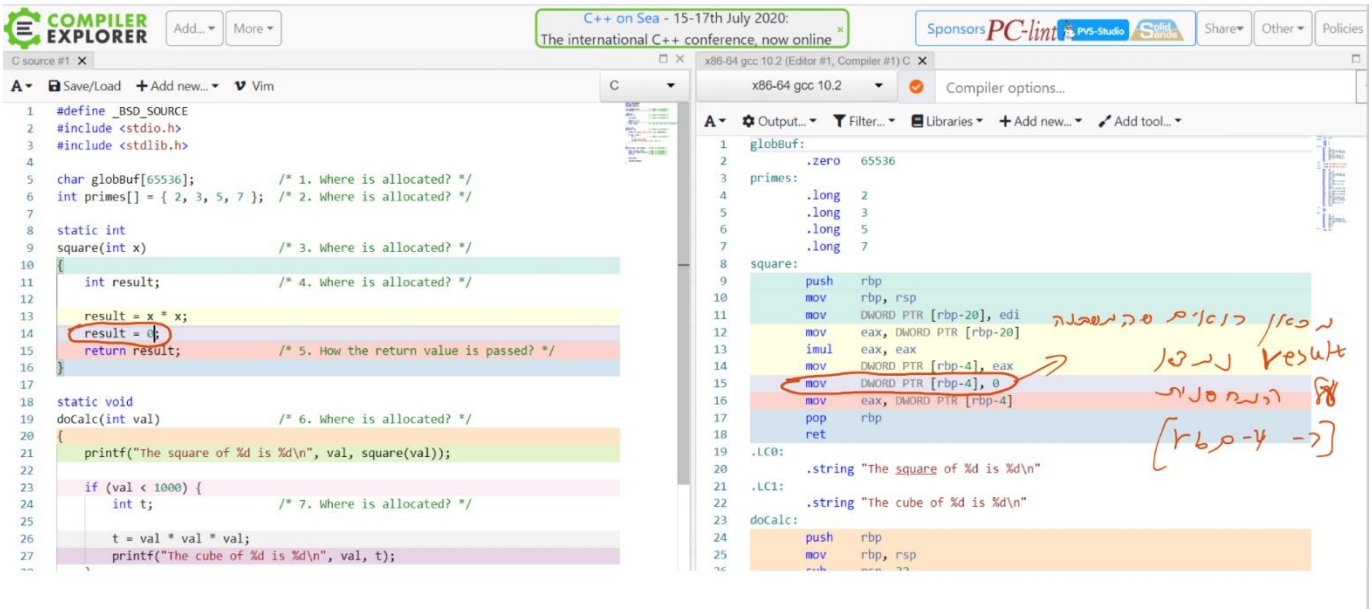
4. המשתנה result נמצא ב stack.

ניתן לראות מהצילום הבא שהשורה result=x*x מתבצעת והתוצאה נשמרת בסופו של דבר על המחסנית בהיסט של 4- מתחילת ה frame של המחסנית של הפונקציה ולכן מוכח מכאן שהמשתנה result נמצא על המחסנית.



בס"ד

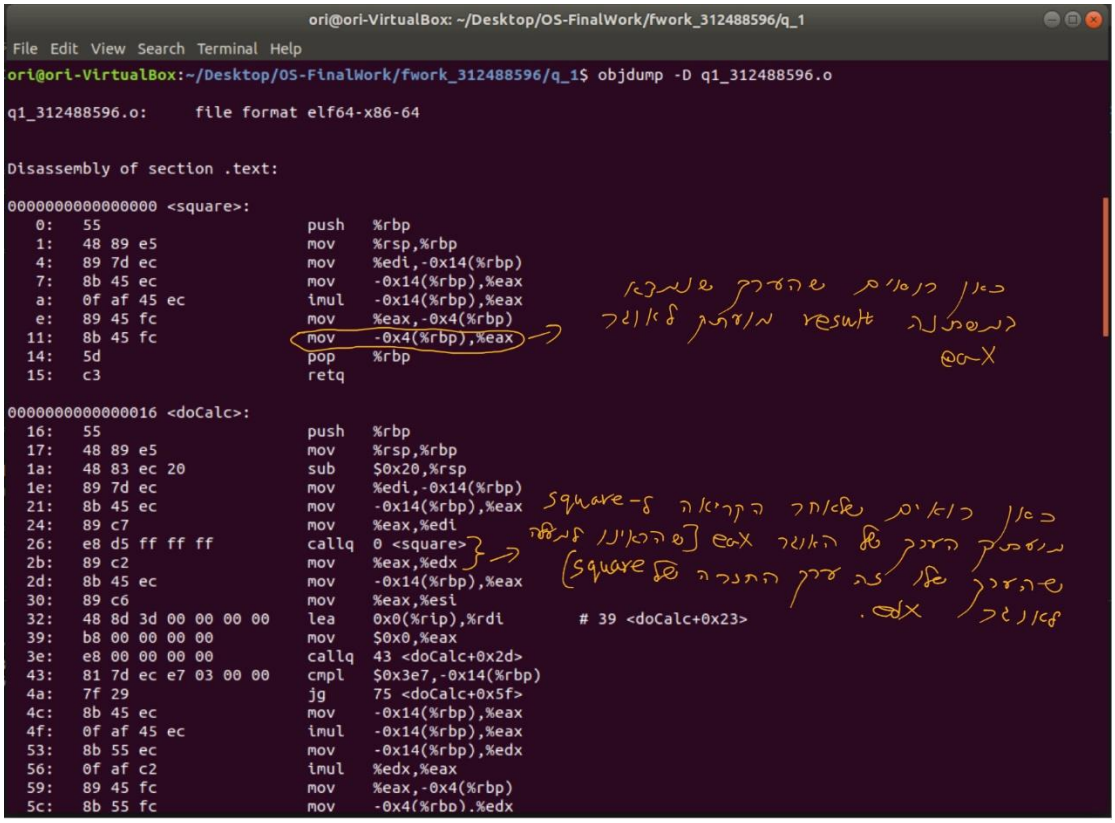
בצילום הבא ניתן לראות את קוד האסמבלי של התכנית (מהאתר godbolt) עם הוספה של שורה 14 (result = 0), ניתן לראות מקוד האסמבלי של שורה זו שהמשתנה result נמצא על המחסנית במיקום rbp-4.




```
return result;          /* 5. How the return value is passed? */
```

5. ערך החזרה מועבר על-ידי רגיסטר.

בצילום הבא של קוד האסמבלי של התכנית ניתן לראות שהערך שנמצא במשתנה result (בפונקציה square) מועתק לאוגר eax, כמו כן בקוד האסמבלי של הפונקציה doCalc ניתן לראות שלאחר הקריאה לפונקציה square מועתק הערך שנמצא באוגר eax שכמו שאמרנו בהתחלה הוא מכיל את הערך של result שזה ערך החזרה של הפונקציה square.



בס"ד

```c
static void doCalc(int val)        /* 6. Where is allocated? */
```

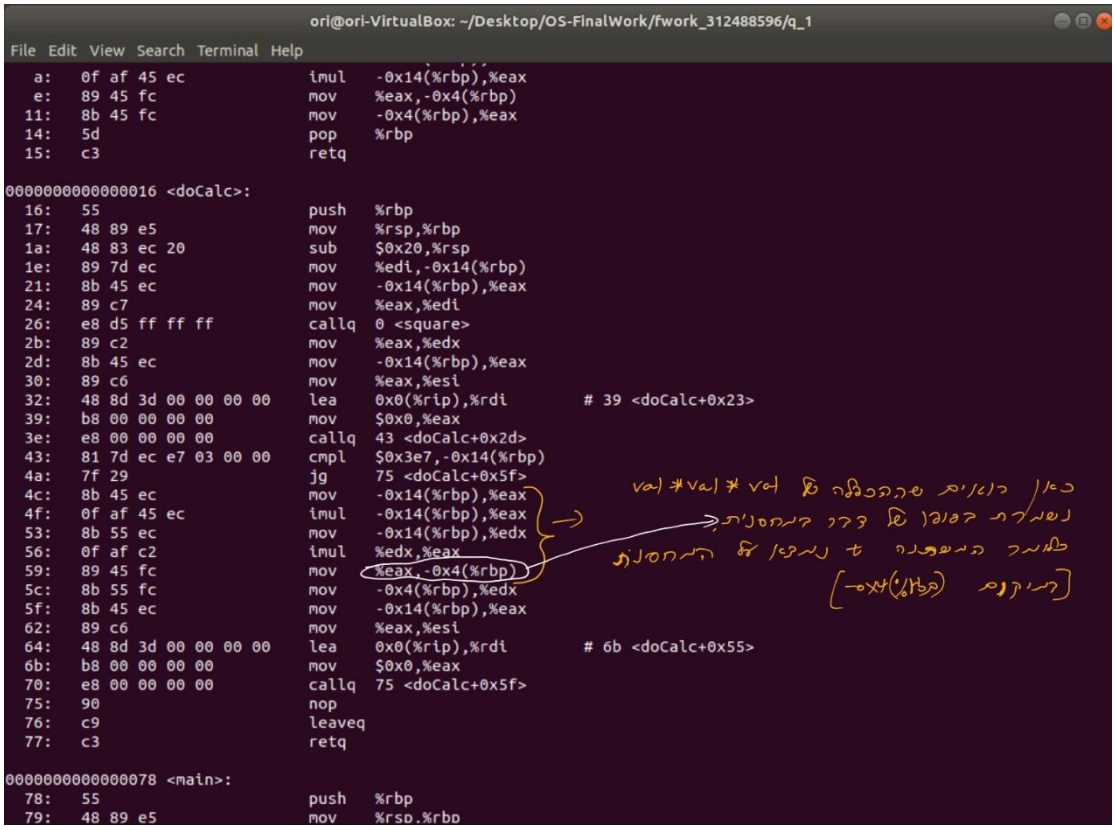
6. הפונקציה doCalc נמצא ב text segment.

לפי מבנה הזכרון של תכניות בשפת c, הוראות של פונקציות נמצאות ב text segment.

```c
int t;                  /* 7. Where is allocated? */
```

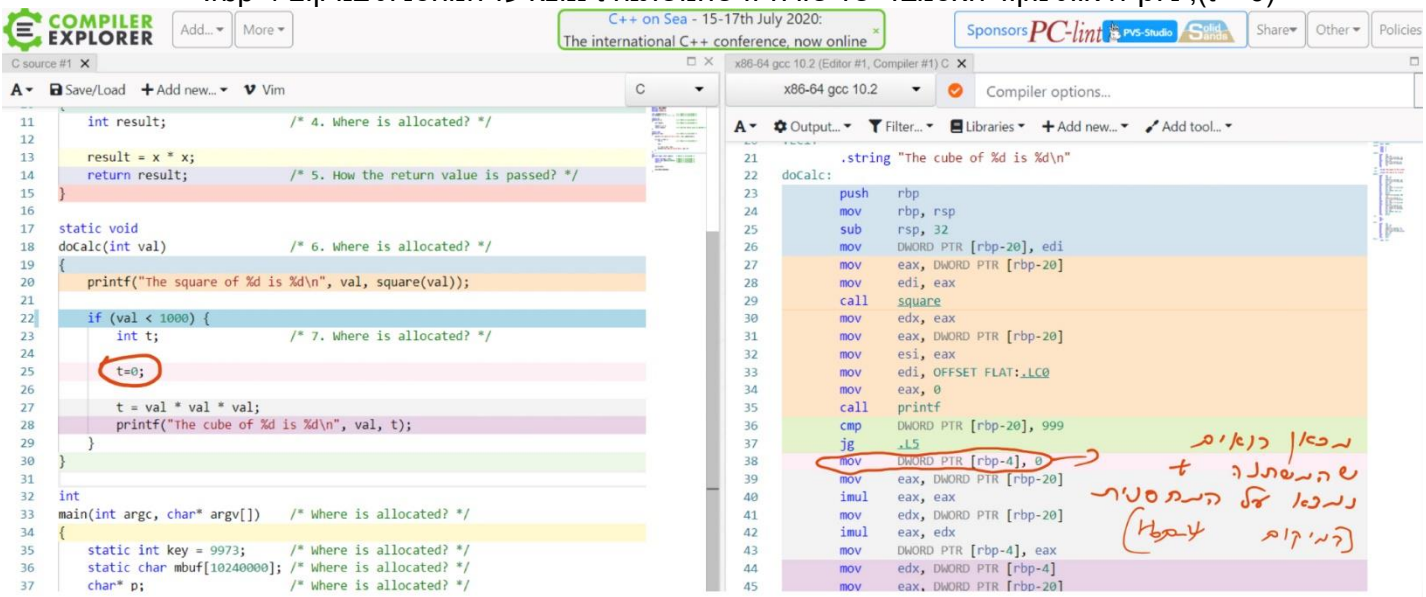
7. המשתנה t נמצא ב stack.

בצילום הבא ניתן לראות שהשורה t=val*val*val מתבצעת והתוצאה נשמרת בסופו של דבר על המחסנית בהיסט של 4- מתחילת ה frame של המחסנית של הפונקציה ולכן מוכח מכאן שהמשתנה t נמצא על המחסנית.



בס"ד

בצילום הבא ניתן לראות את קוד האסמבלי של התכנית (מהאתר godbolt) עם הוספה של שורה 25 (t = 0), ניתן לראות מקוד האסמבלי של שורה זו שהמשתנה t נמצא על המחסנית במיקום rbp-4.




```c
int main(int argc, char* argv[])     /* 8. Where is allocated? */
```

8. הפונקציה main נמצאת ב text segment.

לפי מבנה הזכרון של תכניות בשפת c, הוראות של פונקציות נמצאות ב text segment.

```c
static int key = 9973;       /* 9. Where is allocated? */
```

9. המשתנה key נמצא ב initialized data segment.

לפי מבנה הזכרון של תכניות בשפת c, משתנים סטטיים יכולים להימצא או ב uninitialized data segment או ב initialized data segment כאשר המשתנה ממוקם בחלק הראשון אם המתכנת לא איתחל אותו ובחלק השני כאשר המתכנת איתחל את המשתנה.

מאחר ו key אותחל אז הוא ממוקם ב initialized data segment.

*מופיעים בסוף הקובץ צילומי מסך שמאשרים קביעה זו.

```c
static char mbuf[10240000]; /* 10. Where is allocated? */
```

10. המשתנה mbuf נמצא ב uninitialized data segment (bss).

לפי מבנה הזכרון של תכניות בשפת c, משתנים סטטיים יכולים להימצא או ב uninitialized data segment או ב initialized data segment כאשר המשתנה ממוקם בחלק הראשון אם המתכנת לא איתחל אותו ובחלק השני כאשר המתכנת איתחל את המשתנה.

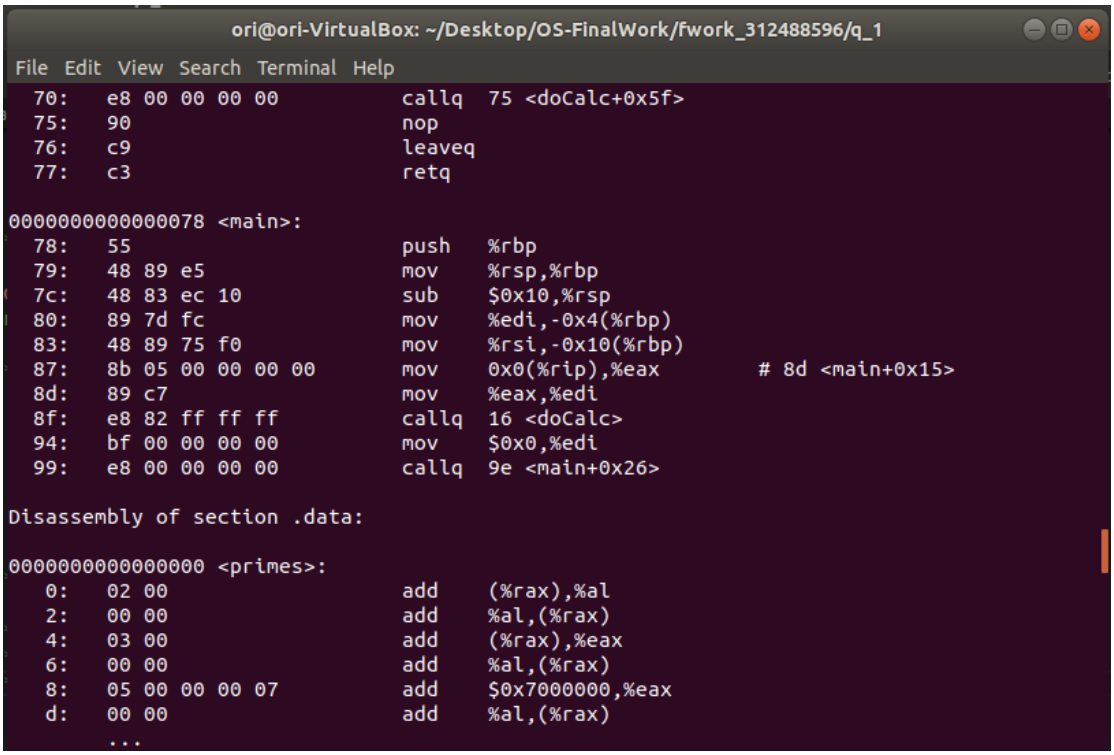
מאחר ו mbuf לא אותחל אז הוא ממוקם ב uninitialized data segment.

*מופיעים בסוף הקובץ צילומי מסך שמאשרים קביעה זו.

```
char* p;                    /* 11. Where is allocated? */
```

11. המשתנה p לא ממוקם בשום אזור זכרון.
תחילה נשים לב ש p הינו משתנה מקומי ששייך ל main ולכן אינו יכול להיות שייך ל initialized data segment ול uninitialized data segment (מכיוון שרק משתנים סטטיים וגלובליים ממוקמים שם) ול text segment (כי רק פקודות ממוקמות שם), בנוסף ניתן לראות ש p לא מופיע בפלטים של הפקודות שמופיעים בסוף הקובץ שמראים את מה שנמצא באזורי הזכרון הללו.
כמו כן p לא נמצא על ה heap מאחר ולא התבצעה הקצאת זכרון דינמית, לכן האפשרות היחידה שנותרה היא ש p נמצא על המחסנית.
על מנת להראות ש p לא נמצא על המחסנית הוספתי צילום מסך של קוד האסמבלי של ה main פעם אחת עם ההצהרה של המשתנה p (כמו התכנית המקורית) ופעם אחרת ללא הצהרה של המתשנה p, ניתן לראות ששניהם הקודים של האסמבלי זהים לחלוטין ולכן ניתן להסיק מכך שלמשתנה p לא מוקצה כלל זכרון.

הצילום הבא הוא הוא הוא של קוד האסמבלי של ה main עם ההצרה על המשתנה p.



בס"ד

הצילום הבא הוא של קוד האסמבלי של ה main ללא ההצהרה של המשתנה p,ניתן לראות שהקודים זהים לחלוטין ומכאן ניתן להסיק שהמשתנה p לא נמצא על המחסנית.

```
ori@ori-VirtualBox: ~/Desktop/OS-FinalWork/fwork_312488596/q_1
File  Edit  View  Search  Terminal  Help
  64:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 6b <doCalc+0x55>
  6b:   b8 00 00 00 00          mov    $0x0,%eax
  70:   e8 00 00 00 00          callq  75 <doCalc+0x5f>
  75:   90                      nop
  76:   c9                      leaveq
  77:   c3                      retq

0000000000000078 <main>:
  78:   55                      push   %rbp
  79:   48 89 e5                mov    %rsp,%rbp
  7c:   48 83 ec 10             sub    $0x10,%rsp
  80:   89 7d fc                mov    %edi,-0x4(%rbp)
  83:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
  87:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 8d <main+0x15>
  8d:   89 c7                   mov    %eax,%edi
  8f:   e8 82 ff ff ff          callq  16 <doCalc>
  94:   bf 00 00 00 00          mov    $0x0,%edi
  99:   e8 00 00 00 00          callq  9e <main+0x26>

Disassembly of section .data:

0000000000000000 <primes>:
   0:   02 00                   add    (%rax),%al
   2:   00 00                   add    %al,(%rax)
   4:   03 00                   add    (%rax),%eax
   6:   00 00                   add    %al,(%rax)
   8:   05 00 00 00 07          add    $0x7000000,%eax
```

בסופו של דבר קיבלנו שהמשתנה p לא נמצא באף אזור זכרון.

מצורפים כאן צילומי מסך של שימוש בכלים בלינוקס שמאשרים את התשובות שלי עבור שאלות 1,2,3,6,8,9,10:

בתמונה א ניתן לראות את הפלט של הפקודה nm -n על הקובץ המקומפל של התכנית, עבור כל אחד מהשמות של המשתנים/הפונקציות שמופיעות בפלט מופיעה אות באנגלית שמציינת את המקום שבו נמצא המשתנה/הפונקציה בזיכרון (על פי ה man) –

b/C – uninitialized data section

D/d – initialized data section

T/t – text section

א.

```
ori@ori-VirtualBox: ~/Desktop/OS-FinalWork/fwork_312488596/q_1
File  Edit  View  Search  Terminal  Help
ori@ori-VirtualBox:~/Desktop/OS-FinalWork/fwork_312488596/q_1$ nm -n q1_312488596.o
                 U exit
                 U _GLOBAL_OFFSET_TABLE_
                 U printf
0000000000000000 b mbuf.2776
0000000000000000 D primes
0000000000000000 t square
0000000000000010 d key.2775
0000000000000016 t doCalc
0000000000000078 T main
0000000000010000 C globBuf
ori@ori-VirtualBox:~/Desktop/OS-FinalWork/fwork_312488596/q_1$
```

בס"ד

בתמונה ב ניתן לראות את הפלט של הפקודה objdump -x על הקובץ המקומפל של התוכנית, עבור כל אחד מהשמות של המשתנים/הפונקציות שמופיעות בפלט (square, doCalc, key, mbuf, globBuf, primes, main) מופיע בפירוש המקום שבו נמצא המשתנה/הפונקציה בזיכרון –

ב.

```
ori@ori-VirtualBox: ~/Desktop/OS-FinalWork/fwork_312488596/q_1
File  Edit  View  Search  Terminal  Help
ori@ori-VirtualBox:~/Desktop/OS-FinalWork/fwork_312488596/q_1$ objdump  -x q1_312488596.o

q1_312488596.o:     file format elf64-x86-64
q1_312488596.o
architecture: i386:x86-64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         0000009e  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000014  0000000000000000  0000000000000000  000000e0  2**4
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          009c4000  0000000000000000  0000000000000000  00000100  2**5
                  ALLOC
  3 .rodata       0000002e  0000000000000000  0000000000000000  00000100  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      0000002a  0000000000000000  0000000000000000  0000012e  2**0
                  CONTENTS, READONLY
  5 .note.GNU-stack 00000000  0000000000000000  0000000000000000  00000158  2**0
                  CONTENTS, READONLY
  6 .eh_frame     00000078  0000000000000000  0000000000000000  00000158  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 q1_312488596.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l     F .text  0000000000000016 square
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000016 l     F .text  0000000000000062 doCalc
0000000000000010 l     O .data  0000000000000004 key.2775
0000000000000000 l     O .bss   00000000009c4000 mbuf.2776
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000010000       O *COM*  0000000000000020 globBuf
0000000000000000 g     O .data  0000000000000010 primes
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 printf
0000000000000078 g     F .text  0000000000000026 main
0000000000000000         *UND*  0000000000000000 exit
```