שאלה 1:

- **קימפול עם gcc:** gcc -g -o pl_q process_layout_q.c
- **הרצת objdump :** objdump -j .text -D –line-number -M intel pl_q
- **הרצת nm:** nm –line-number pl_q

1. `char globBuf[65536];  /* 1. Where is allocated? Uninitialized data segment (bss) */`

בשורה 5, המשתנה globBuf מוקצה על הbss. כלומר על הuninitialized data segment. באזור זה נשמרים משתנים גלובליים וסטטיים שמאותחלים ל-0 או שאינם מאותחלים בקוד התוכנית. המשתנה globBuf הוא משתנה גלובלי ולא אתחלנו אותו ולכן הוקצה שם.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהמשתנה globBuf שהוקצה בשורה 5 בתוכנית, הוקצה על B, כלומר על הbss.

```
0000000000bc5060 B globBuf       /home/noa/Desktop/os/final/process_layout_q.c:5
                 w __gmon_start__
```

מצרפת צילום מסך מאתר man המסביר את הפירוש של האות B:

```
B
b
        The symbol is in the uninitialized data section (known as BSS).
```

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

בשורה 6, המערך primes מוקצה על ה Initialized data segment. באזור זה נשמרים משתנים גלובליים וסטטיים שמאותחלים בקוד התוכנית. נשים לב שבתוכנית שלנו אתחלנו את המערך עם ערכים ולכן הוא נשמר שם.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהמשתנה primes שהוקצה בשורה 6 בתוכנית, הוקצה על D, כלומר על ה initialized data segment.

```
0000000000201010 D primes        /home/noa/Desktop/os/final/process_layout_q.c:6
```

מצרפת צילום מסך מאתר man המסביר את הפירוש של האות D:

```
D
d
        The symbol is in the initialized data section.
```

.3

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

בשורה 9, הפונקציה square מוקצית ב text (code) segment. (נשים לב אבל שהמשתנה x מוקצה על הstack.) Text Segment זהו אזור המכיל העתק של אוסף הפקודות של התוכנית. בזמן הרצת התוכנית, כתובת הפקודה הבאה לביצוע מצויה באוגר הנקרא program counter.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהפונקציה square שהוקצתה בשורה 9 בתוכנית, הוקצתה על t, כלומר על ה Text segment.

```
000000000000068a t square        /home/noa/Desktop/os/final/process_layout_q.c:9
```

מצרפת צילום מסך מאתר man המסביר את הפירוש של האות t:

```
T
t
```

The symbol is in the text (code) section.

.4

```
int result;                /* 4. Where is allocated? Stack frame for square()  */
```

בשורה 11, המשתנה result מוקצה בstack של הפונקציה square.

ניתן לראות לפי השורות המצורפות משימוש **בobjdump** שהוקצו מצביעים לstack frame של הפונקציה square ולאחר מכן הוקצה מקום בשביל המשתנה x (עם הערך edi שהתקבל בפונקציה). הערך של x מועבר אל eax, ואז מתבצעת ההכפלה. התוצאה מושמת במשתנה result הנשמר בזיכרון בstack במיקום rbp-0x4.

```
000000000000068a <square>:
square():
 68a:   55                      push   rbp
 68b:   48 89 e5                mov    rbp,rsp
 68e:   89 7d ec                mov    DWORD PTR [rbp-0x14],edi
 691:   8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
 694:   0f af 45 ec             imul   eax,DWORD PTR [rbp-0x14]
 698:   89 45 fc                mov    DWORD PTR [rbp-0x4],eax
 69b:   8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
 69e:   5d                      pop    rbp
 69f:   c3                      ret
```

הסבר: בשורה הראשונה rbp מצביע לסוף הstack frame הקודם. לאחר מכן מוקצה מצביע חדש rsp המצביע לתחילת ה stack frame של הפונקציה square. ואז מעתיק את הערך של rsp לrbp ושניהם מצביעים לתחילת הstack frame של הפונקציה. בשורה 68e ניתן לראות שבמחסנית בכתובת rbp-0x14 מוקצה המשתנה x על הstack frame עם הערך edi , מתבצעת ההכפלה (שורה 694) והתוצאה מאוחסנת בresult במיקום rbp-0x4 (שורה 698). נשים לב, שההקצאה נעשית רק לאחר השמת ערך לresult (שורה 13 בתוכנית C) ולא בשורת ההצהרה שלו (שורה 11 בתוכנית C).

.5

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

בשורה 14, המשתנה result מוחזר מהפונקציה דרך הregister.

ניתן לראות זאת בשורות המצורפת ע"י שימוש ב**odjdump:**

```
/home/noa/Desktop/os/final/process_layout_q.c:14
 69b:   8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
/home/noa/Desktop/os/final/process_layout_q.c:15
 69e:   5d                      pop    rbp
 69f:   c3                      ret
```

<u>הסבר:</u> אנו שמים לב שלאחר החישוב על result התוצאה מועברת מהמחסנית אל ה eax register וזהו הערך המוחזר מהפונקציה. לאחר מכן הrbp register קופץ להצביע לכתובת שממנה קראנו לפונקציה ונשמרה על המחסנית כאשר קראנו לפונקציה (עם call) ולשם הערך מוחזר בעזרת rbp.

.6

```
doCalc(int val)                /* 6. Where is allocated? Text segment */
```

בשורה 18, הפונקציה doCalc מוקצית ב text (code) segment. (באופן דומה לפונקציה square)

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהפונקציה doCalc שהוקצתה בשורה 18 בתוכנית, הוקצתה על t, כלומר על ה Text segment.

```
00000000000006a0 t doCalc        /home/noa/Desktop/os/final/process_layout_q.c:18
```

.7

```
int t;                    /* 7. Where is allocated? Stack frame for doCalc()*/
```

בשורה 23, המשתנה t מוקצה בstack של הפונקציה doCalc.

ניתן לראות לפי השורות המצורפות משימוש **בobjdump** שהוקצו מצביעים לstack frame של הפונקציה doCalc ולאחר מכן הוקצה מקום בשביל המשתנים val ו- t .
נשים לב, שההקצאה של t תלויה בתנאי if (בשורה 25 בתוכנית). במקרה ש val<1000 אז t מוקצה על המחסנית בכתובת rbp-0x4 (בשורה 6e3). הקומפיילר מקצה משתנים על המחסנית רק אם הם מאותחלים ולכן ההקצאה מתבצעת רק כאשר מושם בו הערך (בשורה 25 בתוכנית ולא בשורה 23).
כמובן שאם התנאי בif לא מתקיים t לא מוקצה בזיכרון (ניתן לראות בשורה 6d4 שאם לא מתקיים תתבצע קפיצה לשורה 6ff).

```
00000000000006a0 <doCalc>:
doCalc():
 6a0:   55                      push   rbp
 6a1:   48 89 e5                mov    rbp,rsp
 6a4:   48 83 ec 20             sub    rsp,0x20
 6a8:   89 7d ec                mov    DWORD PTR [rbp-0x14],edi
 6ab:   8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
 6ae:   89 c7                   mov    edi,eax
 6b0:   e8 d5 ff ff ff          call   68a <square>
 6b5:   89 c2                   mov    edx,eax
 6b7:   8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
 6ba:   89 c6                   mov    esi,eax
 6bc:   48 8d 3d f1 00 00 00    lea    rdi,[rip+0xf1]        # 7b4
 6c3:   b8 00 00 00 00          mov    eax,0x0
 6c8:   e8 83 fe ff ff          call   550 <printf@plt>
 6cd:   81 7d ec e7 03 00 00    cmp    DWORD PTR [rbp-0x14],0x3e7
 6d4:   7f 29                   jg     6ff <doCalc+0x5f>
 6d6:   8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
 6d9:   0f af 45 ec             imul   eax,DWORD PTR [rbp-0x14]
 6dd:   8b 55 ec                mov    edx,DWORD PTR [rbp-0x14]
 6e0:   0f af c2                imul   eax,edx
 6e3:   89 45 fc                mov    DWORD PTR [rbp-0x4],eax
 6e6:   8b 55 fc                mov    edx,DWORD PTR [rbp-0x4]
```

<u>הסבר:</u> בשורה הראשונה rbp מצביע לסוף הstack frame הקודם. לאחר מכן מוקצה מצביע חדש rsp המצביע לתחילת ה stack frame של הפונקציה doCalc. ואז מעתיק את הערך של rsp לrbp ושניהם מצביעים לתחילת הstack frame של הפונקציה. בשורה השלישית נוצר מקום הנקרא stack frame (בגודל 20 בתים) בו ניתן לשים משתנים מקומיים חדשים של הפונקציה ועליו ניתן להקצות משתנים חדשים. (rsp מצביע לסוף הקטע הזה). לאחר מכן במקום במחסנית בכתובת rbp-0x14 מוקצה המשתנה val על הstack frame עם הערך ב- edi שזהו הערך של שהתקבל בפונקציה. לאחר מכן על הערך הזה במידה ומתקיים תנאי הif עושים את פעולות ההכפלה הרצויות והערך של t מוקצה בהתאם עם התוצאה.

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

בשורה 18, פונקציית main מוקצית ב text (code) segment. (באופן דומה לפונקציות האחרות) הפונקציה מקבלת ערכים מהcommand line.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהפונקציה main שהוקצתה בשורה 31 בתוכנית, הוקצתה על t, כלומר על ה Text segment:

```
0000000000000702 T main /home/noa/Desktop/os/final/process_layout_q.c:31
```

9.
```
static int key = 9973;       /* Where is allocated? Initialized data segment */
```

בשורה 33, מוקצה המשתנה הסטטי key על ה Initialized data segment. באזור זה נשמרים משתנים גלובליים וסטטיים שמאותחלים בקוד התוכנית ומאחר והמשתנה שלנו הוא סטטי ומאותחל עם ערך הוא הוקצה שם.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהמשתנה key שהוקצה בשורה 33 בתוכנית, הוקצה על d, כלומר על ה initialized data segment.

```
0000000000201020 d key.2775
```

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

בשורה 34, מוקצה המערך הסטטי mbuf על הbss, כלומר על ה uninitialized data segment.
באזור זה נשמרים משתנים גלובליים וסטטיים שמאותחלים ל-0 או שאינם מאותחלים בקוד התוכנית.
המשתנה mbuf הוא משתנה **סטטי** ולא אתחלנו אותו ולכן הוקצה שם.

ניתן לראות לפי השורה המצורפת משימוש **בnm** שהמשתנה mbuf שהוקצה בשורה 34 בתוכנית, הוקצה על b, כלומר על הbss.

```
0000000000201060 b mbuf.2776
```

11.
```
char* p;                     /* Where is allocated? Stack frame for main() */
```

בשורה 35, באופן תאורטי המצביע p מוקצה בstack של הmain. נשים לב שבפועל p לא מאותחל ולכן לא מוקצה לו מקום במחסנית ולכן לא ניתן לראות זאת בעזרת הכלים שניתנו לנו.

במחסנית נמצאים המשתנים המקומיים של הפונקציה. עבור כל פונקציה מוגדר stack frame בו לאחר מכן מוגדרים ומאותחלים המשתנים המקומיים שלה. המשתנה p זהו משתנה לוקלי של פונקציית הmain ולכן מוגדר על המחסנית שלה. זהו לא משתנה גלובלי ולא סטטי ולכן לא הוקצה בbss (כמו key או mbuf). בפועל הקומפיילר לא מקצה לו מקום על הstack עדיין כי הוא לא מאותחל, ולכן רק לאחר השמת נתונים בp אנו נראה את ההקצאה שלו (בעזרת objdump למשל) על הstack.