מטלת הגשה בשפת C במבנה מחשבים ספרתיים

שם המגיש: איילון קפל
תעודת זהות: 207807025

B. שאלות חלק תיאורטי:

1. ראשית, ישנו מונח הנקרא scope המתאר את מרחב המשתנים אשר מוקצה בכניסה לפונקציה, לולאה, תנאי וכו', המרחב הזה יהיה נגיש רק עבור המקום שהוא הוגדר בו, לכן לדוגמה אם קיימות שתי לולאות (שאינן אחת בתוך השנייה) ובכל אחת מוגדרים משתנים בעלי אותו שם, כאשר התוכנה רצה המשתנים האלו יהיו שמורים תחת כתובות שונות בזיכרון ובכך למעשה שונים זה מזה, כלומר פונקצייה אחת לא נגישה למרחב המשתנים שפונקצייה אחרת מגדירה, המשתנים שתוארו באופן זה הינם **משתנים לוקאליים**.
זאת בשונה **ממשתנים גלובליים** שהם משתנים שאנו מגדירים אותם להיות נגישים לכל הקוד ובכך תתאפשר לנו גישה אליהם בכל פונקצייה\לולאה\תנאי וכו'.

בקוד הניתן בexample2 ניתן לראות כי המשתנה menu **הינו משתנה גלובלי** מפני שהוא מוגדר בהיררכיה הגבוהה ביותר (ולא בתוך פונקצייה לדוגמה) ולכן מתאפשרת אליו הגישה לכל אורך הקוד כמו לדוגמה בפונקציה main, לסיכום הסקופ של משתנה זה הוא כל התוכנה.
ולעומת זאת נראה כי לדוגמה המשתנה i **הינו משתנה לוקאלי** המוגדר במספר פונקציות אשר כל אחת שומרת אותו בכתובת נפרדת בזיכרון, הסקופ של משתנה זה הינו הפונקצייה שבה הוא מוגדר.

2. לאחר הרצה של התוכנית עד להגעה לbreakpoint בנקודה בה יצרנו את המטריצה Mat ומילאנו אותה בערכים, חלון disassembly מציג עבורה את הערכים הבאים:

Locals

Name	Value	Type
Mat	0x007af7a8 {0x007af7a8 {5, 6, 7, 8, 9, 10, 11, 12, 13, 14}, 0x007af7d...	int[10][10]
[0]	0x007af7a8 {5, 6, 7, 8, 9, 10, 11, 12, 13, 14}	int[10]
[1]	0x007af7d0 {15, 16, 17, 18, 19, 20, 21, 22, 23, 24}	int[10]
[2]	0x007af7f8 {25, 26, 27, 28, 29, 30, 31, 32, 33, 34}	int[10]
[3]	0x007af820 {35, 36, 37, 38, 39, 40, 41, 42, 43, 44}	int[10]
[4]	0x007af848 {45, 46, 47, 48, 49, 50, 51, 52, 53, 54}	int[10]
[5]	0x007af870 {55, 56, 57, 58, 59, 60, 61, 62, 63, 64}	int[10]
[6]	0x007af898 {65, 66, 67, 68, 69, 70, 71, 72, 73, 74}	int[10]
[7]	0x007af8c0 {75, 76, 77, 78, 79, 80, 81, 82, 83, 84}	int[10]
[8]	0x007af8e8 {85, 86, 87, 88, 89, 90, 91, 92, 93, 94}	int[10]
[9]	0x007af910 {95, 96, 97, 98, 99, 0, 1, 2, 3, 4}	int[10]

ניתן לראות כי המטריצה בנויה מ10 מערכים אשר כל אחד מהם מכיל 10 משתנים מסוג int.
הגודל 10 נקבע על ידי הקבוע M.
הכתובת ההתחלתית של המטריצה הינה 0x007af7a8 והאיבר האחרון נגמר בכתובת 0x007af910+28=0x007af938, כלומר, המטריצה הינה בגודל
0x007af938-0x007af7a8=0x938-0x7a8=0x190=400(decimal), כלומר, 400 ביטים.
הסיבה לכך היא שמשתנה מסוג int נשמר על ידי 32 ביטים בזיכרון, כלומר 4 בתים (כאשר כל כתובת בזיכרון מייצגת בית, 8 ביטים), מכיוון שיש 100 משתנים כאלו במטריצה (מטריצה בגודל 10X10) המקום שהיא תופסת בזיכרון הינו אכן 400 ביטים.
3. כפי שנכתב בפורום במודל, יש להתעלם מהסעיף הזה בגלל בעיה בניסוח
4. על מנת לראות את ערך הרגיסטרים או בפרט את רגיסטר PC בהגעה לפקודה הראשונה של הפונקציה compute trace נשים breakpoint בנקודה הזו כפי שניתן לראות בתמונה:

```
99   //-----------------------------------------------------------
100  int ComputeTrace(int Mat[M][M]) {
101      int Trace = 0, i;
102      for (i = 0; i < M; i++) Trace += Mat[i][i];
103      return Trace;
104  }
105  //-----------------------------------------------------------
```

לאחר הגעה לנקודה זו נראה כי:

```
Disassembly   ConsoleApplication1.c
Address: ComputeTrace(int[10] *)
Viewing Options
009E178C  lea         edi,[ebp-0D8h]
009E1792  mov         ecx,36h
009E1797  mov         eax,0CCCCCCCCh
009E179C  rep stos    dword ptr es:[edi]
009E179E  mov         ecx,offset _A35DBDDC_consoleapplication1@c (09EC003h)
009E17A3  call        @__CheckForDebuggerJustMyCode@4 (09E122Bh)
    int Trace = 0, i;
009E17A8  mov         dword ptr [Trace],0
    for (i = 0; i < M; i++) Trace += Mat[i][i];
009E17AF  mov         dword ptr [i],0
009E17B6  jmp         ComputeTrace+41h (09E17C1h)
009E17B8  mov         eax,dword ptr [i]
009E17BB  add         eax,1
009E17BE  mov         dword ptr [i],eax

Registers
EAX = 009EC003 EBX = 00AE5000 ECX = 009EC003 EDX = 00000001 ESI = 0086F964 EDI = 0086F958
  EIP = 009E17A8 ESP = 0086F874 EBP = 0086F958 EFL = 00000246

0x0086F950 = CCCCCCCC

Call Stack
Name        Lang
ConsoleAp...  C
ConsoleAp...  C
[External Co...
```

הפקודה הראשונה בהגעה לפונקציה בשפת assembly נמצאת בכתובת 0x009E17A8, ערך זה נשמר ברגיסטר EIP-Extended Instruction Pointer השקול לרגיסטר PC-Program Counter והוא מחזיק את הכתובת של הפקודה שאותה יש לבצע כעת.

5. לאחר השמה של breakpoint בהגעה לפונקצייה fillmatrix והרצה של התוכנית באופן כזה אשר קורא לפונקציה אנו רואים כי הפונקציה נמצאת בכתובת הבאה: 0x00F61930

```
--- c:\users\eylonk\source\repos\consoleapplication3\consoleapplication3\consoleapplication3.c
//-----------------------------------------------------------
//                  Fill Matrix
//-----------------------------------------------------------
void FillMatrix(unsigned int Mat[M][M], int offset) {
00F61930  push        ebp
```

כמו כן, הכתובת בה הפונקצייה מסתיימת הינה: 0x00F619AD כאשר הפקודה return גוררת הפעלה של מספר פקודות pop אשר ביציאה מהפונקצייה מחזירות את המידע אשר נדחף לרגיסטרים בעת הכניסה לפונקצייה.

```
00F619A9  jmp         FillMatrix+31h (0F61961h)
    return;
}
00F619AB  pop         edi
00F619AC  pop         esi
00F619AD  pop         ebx
```

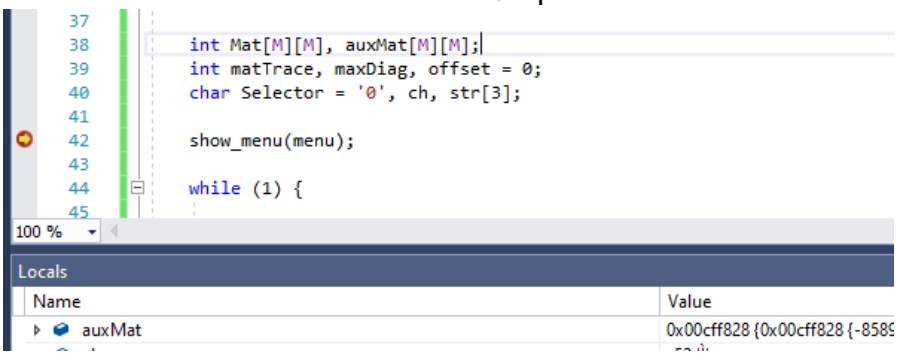
מכאן אנו מבינים כי גודל הפונקציה הינו 125 בתים, סוג הזיכרון שבו התוכנית שמורה במהלך הריצה הינו RAM מפני שכאשר אנו נכנסים לפונקצייה, נוצר scope חדש שבו מוגדרים משתנים חדשים ונעשים חישובים אשר הינם זמניים, לאחר ביצוע הפונקצייה התוצאות של הפעולות שהיא יכלה לבצע

הן עדכון של ערכים בזיכרון (לדוגמה כתיבה של ערכים שונים למטריצה אשר שמורה במקום נתון בזיכרון והפונקצייה קיבלה את המקום הזה כפויינטר) או החזרת ערך כלשהו, לאחר מכן כל המידע לגבי הפונקצייה הכלול במשתנים שהיא הגדירה וכו' אינו רלוונטי יותר ולכן אנו רוצים שהוא יימחק, מסיבה זו הזיכרון של הפונקצייה הינו RAM שכן זהו זיכרון נדיף ולכן השימושים בו הם לחישובים זמניים, בניגוד לתוכנית עצמה אשר שמורה במקום קבוע בזיכרון (ואינה תימחק) מכיוון שהיא מכילה את המידע על הפקודות שיש לבצע ומידע זה חייב להיות זמין לנו תמיד ולכן לא יכול להיות זמני.

כמו כן, הקשר בין הפעולות שמבצעת הפונקצייה לבין השם שלה הן שהפונקצייה מקבלת פויינטר למטריצה אשר קיימת בזיכרון ובנוסף לכך מקבלת ערך offset ובאמצעות כך מחשבת ערכים בעלי חוקיות כלשהי התלויה בoffset ומזינה אותם באיברי המטריצה, מכאן השם של הפונקצייה fillmatrix שכן הפונקצייה "ממלאת" את המטריצה בערכים ובכך נוכל לבצע פעולות על המטריצה כאשר יש בה ערכים כלשהם (לעומת לפני הקריאה לפונקצייה בפעם הראשונה בה במטריצה לא הוגדרו שום ערכים).

6. המשתנה auxmat מוגדר בתחילת הפונקצייה main ולכן למעשה זהו הscope שלו, כלומר, כל לולאה, תנאי, פונקצייה וכו' אשר מבוצעות במהלך הפונקצייה main (וגורמות לפתיחת scope נוסף) נגישות עדיין למשתנה.

נראה כי הכתובת בה הוא הוגדר בזיכרון בעת ריצת התוכנית הינה : 0x00cff828



7. לאחר הוספת קטע הקוד לתוכנית שאנו מריצים נוכל להסתכל בתרגום של הקומפיילר עבורה לשפת assembly ונראה כי התוצאה הינה:

```
    a = a > b ? a : b;
00101D07  mov         eax,dword ptr [a]
00101D0D  cmp         eax,dword ptr [b]
00101D13  jle         main+73h (0101D23h)
00101D15  mov         ecx,dword ptr [a]
00101D1B  mov         dword ptr [ebp-518h],ecx
00101D21  jmp         main+7Fh (0101D2Fh)
00101D23  mov         edx,dword ptr [b]
00101D29  mov         dword ptr [ebp-518h],edx
00101D2F  mov         eax,dword ptr [ebp-518h]
00101D35  mov         dword ptr [a],eax
```

ניתן לראות כי שורת הקוד מתורגמת למספר פעולות בסיסיות יותר בשפת assembly כמתואר בתמונה, בין היתר הפעולות הינן הזזות של ערכים לרגיסטרים על מנת לעשות פעולות, קפיצה למקומות בזיכרון כתלות בתוצאות חישוב מכיוון שקיים תנאי שלפיו נעשית בדיקה האם a>b וכתוצאה של הבדיקה הזו אנו נבצע אחת מבין שתי פעולות שונות ועוד פעולות שבוצעו על מנת לממש את שורת הקוד.