בס"ד

Where is allocated?

1. line 5: char globBuf[65536];
Answer: uninitialized data segment

2.line 6: int primes[] = { 2, 3, 5, 7 };
Answer: initialized data segment

3.line 9: square(int x)
Answer: allocated in frame for square()

4.line 11: int result;
Answer: allocated in frame for square()

Q5: How the return value is passed?
5.line 14: return result;
Answer: return value is passed via register

6.line 18: doCalc(int val)
Answer: allocated in frame for doCalc()

7.line 23: int t;
Answer: allocated in frame for doCalc()

8.line 31: main(int argc, char* argv[])
Answer: allocated in frame for main()

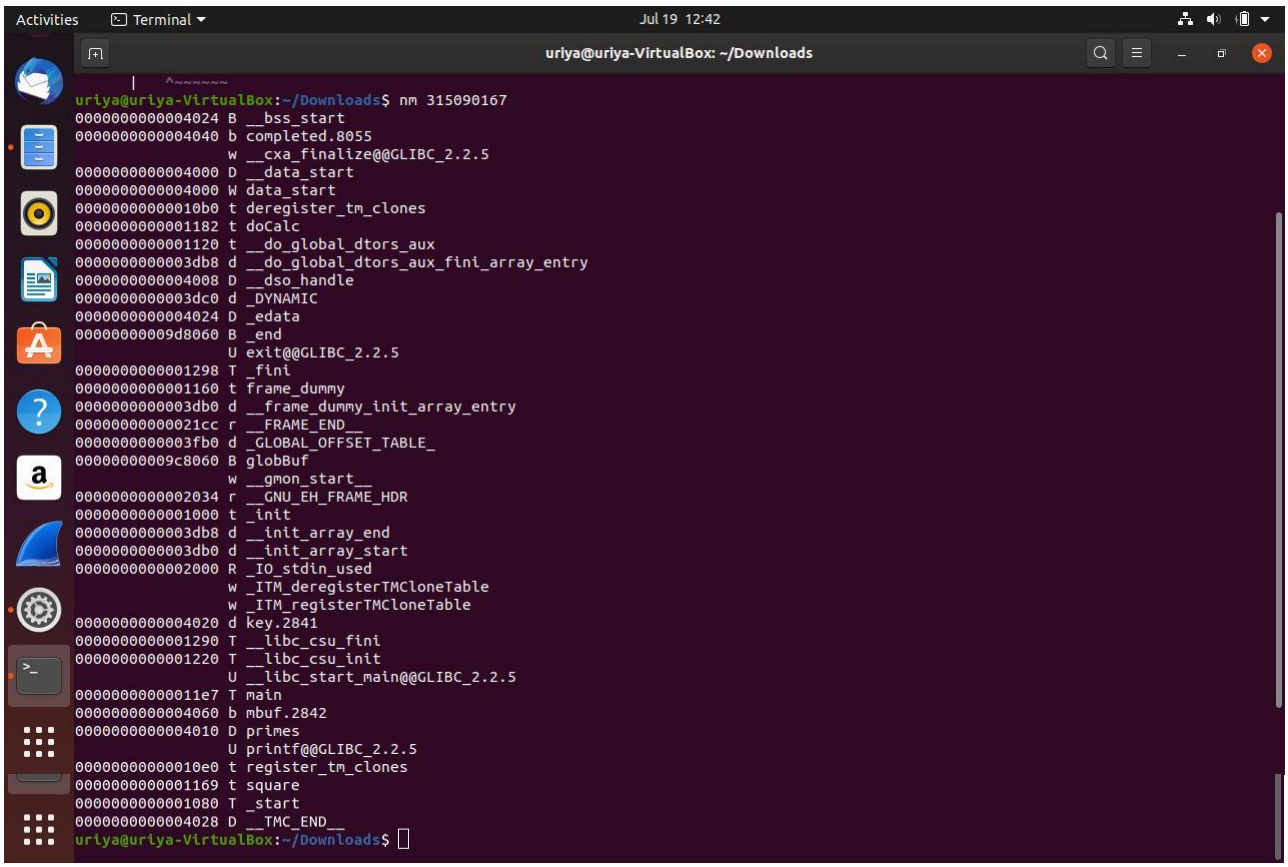
9.line 33: static int key = 9973;
Answer: initialized data segment

10. line 34: static char mbuf[10240000];
Answer: uninitialized data segment

11.line 35: char* p;
Answer: allocated in frame for main()

הפקודה nm (פקודה על object files) נותנת לנו רשימה של הסמלים המופיעים בקובץ. הרצתי את הפקודה הזו על הקובץ שלנו וקיבלתי את הפלט הבא:

בס"ד



ניתן לראות כי הפקודה מבילה לנו את הדברים הבאים:

1 . ערך הסמל בהקסדצימלי (ברירת מחדל וניתן לשנות).

2. סוג הסמל – יבואר למטה על חלק מהסמלים בהם השתמשתי לפתרון השאלה , **<u>דגש</u>**: אם הסמל הוא באות גדולה אזי הסמל הוא סמל גלובלי , באות קטנה סמל מקומי.

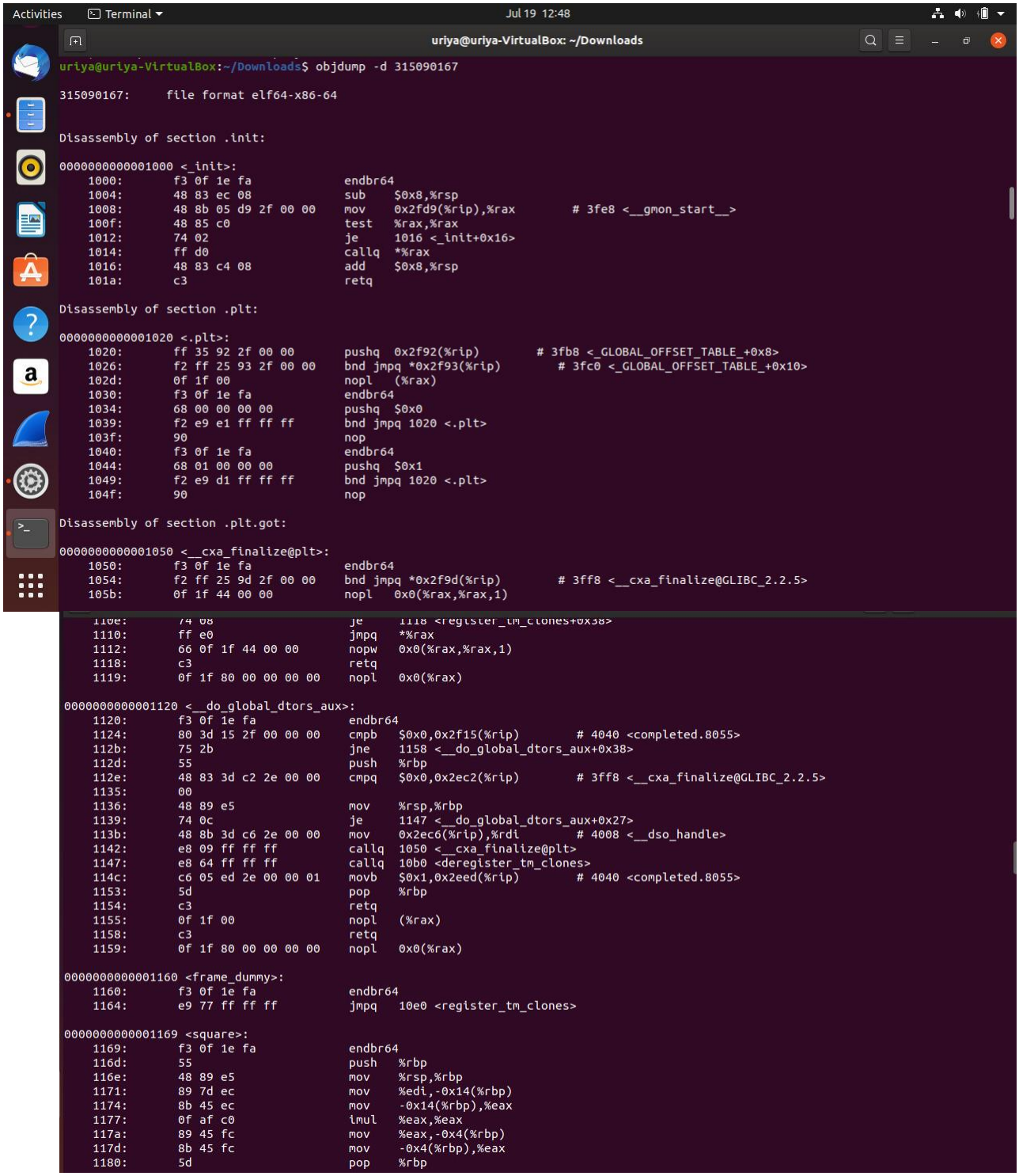
- בסמלים **globBuf** ו-**mbuf** מופיעות האות **'b'\'B'** המציינות שהסמל הזה נמצא ב **uninitialized data.**
  תשובה לשאלות 1 ו- 10.

- בסמלים **primes** ו-**key** מופיעות האותיות **'d'\'D'** המציינות שהסמל נמצא ב **initialized data.**
  תשובה לשאלות 2 ו-9.

- בסמלים **main**, **doCalc**, **square** מופיעים האותיות **'t'** \ **'T'** המציינות שהסמלים הללו נמצאים ב **text section**, כלומר נוצר frame חדש עבורם.
  תשובה לשאלות 3, 6 ו-8.

הפקודה objdump -d נותנת לנו את הקובץ מחולק ל sections עם קוד אסמבלי המביע את הפעולות המתבצעות בכל section.
לאחר הרצת הפקודה קיבלנו את הפלט הבא:

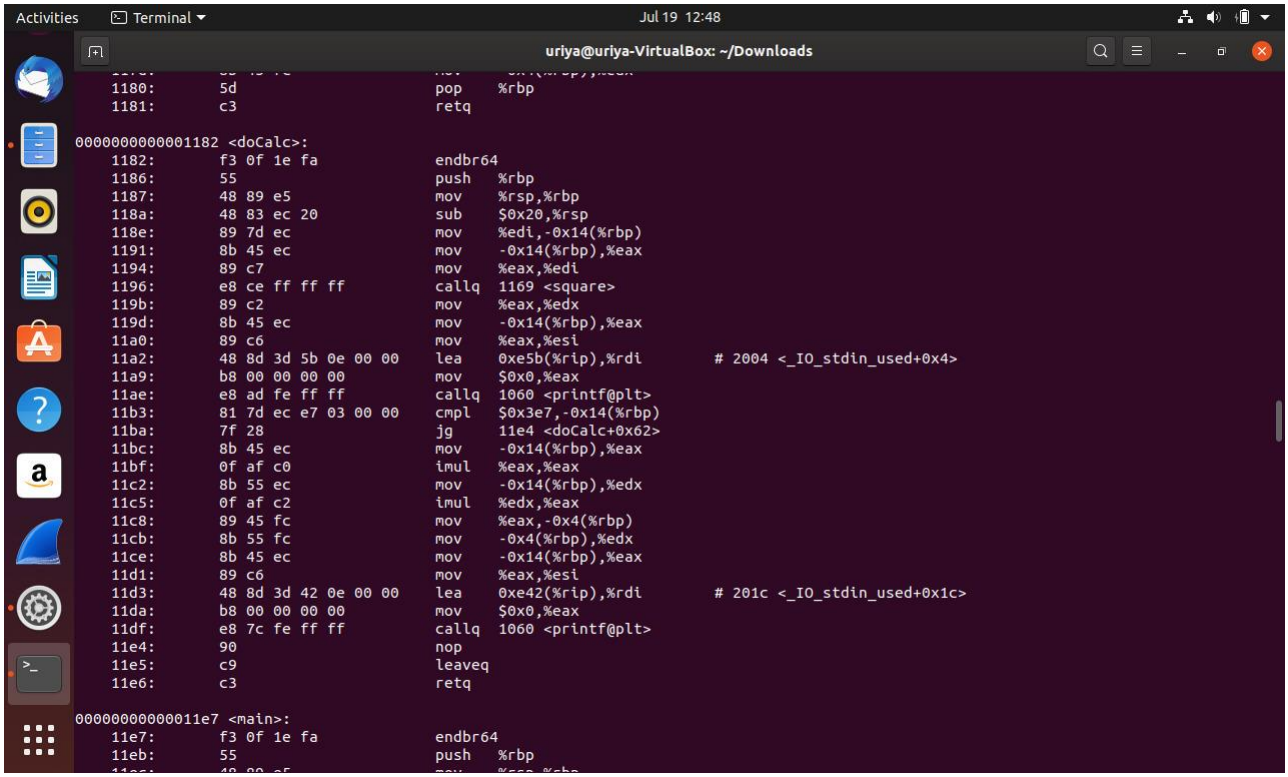
בס"ד



ניתן לראות שבתוך הframe של square אנו דוחפים את result לתוך rbp (שהוא רגיסטר), ובסוף הפונקציה אנחנו מחזירים אותו באמצעות pop , ולבן result מועבר ע"י רגיסטר.

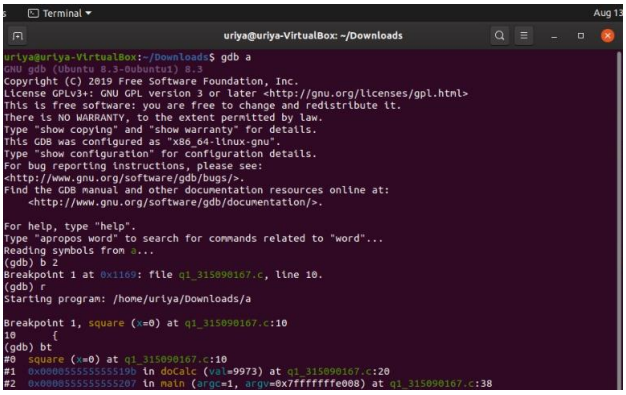
** למדנו כי לכל frame יש מחסנית וזיכרון שמוקצה לו על ידי המערכת , והזיכרון הנ"ל משוחרר לאחר סיום הפונקציה (frame) נגמר. ניתן לראות כי בכל frame למשל ב docalc, , man , square שאם הגדרנו משתנה בתוך אותו frame מתבצעת הפקודה push- דחיפה של המשתנה למחסנית של ה frame, אותו משתנה ישתחרר בסיום ה frame. (תשובה לשאלות 4 , 7 ,11).

בס"ד



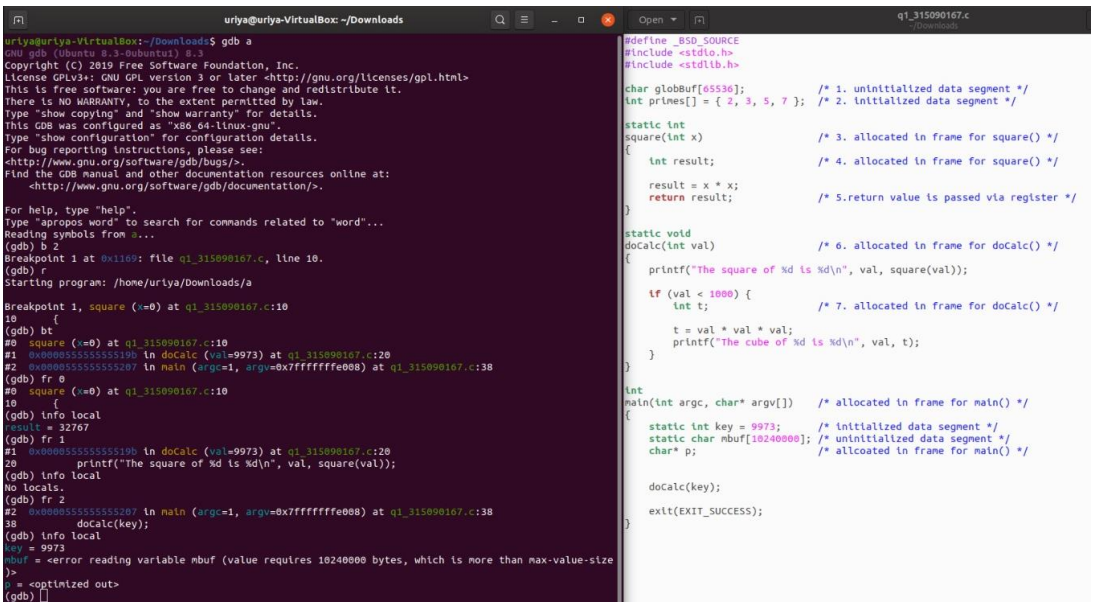
** ניתן לפתור שאלה זו גם באמצעות שימוש בפקודה size, כאשר כל פעם נוסיף עוד שורת קוד מהקוד המקורי שקיבלנו ונריץ את פקודת size, ואז נדע בכל הרצה לאיפה נוסף המשתנה החדש.

פתרון נוסף לשאלה זו הוא באמצעות שימוש ב gdb , אנו נראה כמה דוגמאות כיצד לעשות זאת.
נשתמש בפקודה bt(backtrace) המראה לנו מיעד לגבי הקוד שלנו – כמה פריימים יש לנו בקוד.



כפי שניתן לראות יש לנו 3 פריימים: פריים 0 – של square, פריים 1- של doCalc, פריים 2 – של main.
בבדי לדעת אילו משתנים יש בתוך כל פריים נשתמש בפקודה info locals המראה לנו את המשתנים המקומיים וערכם בתוך הפריים.

בס"ד

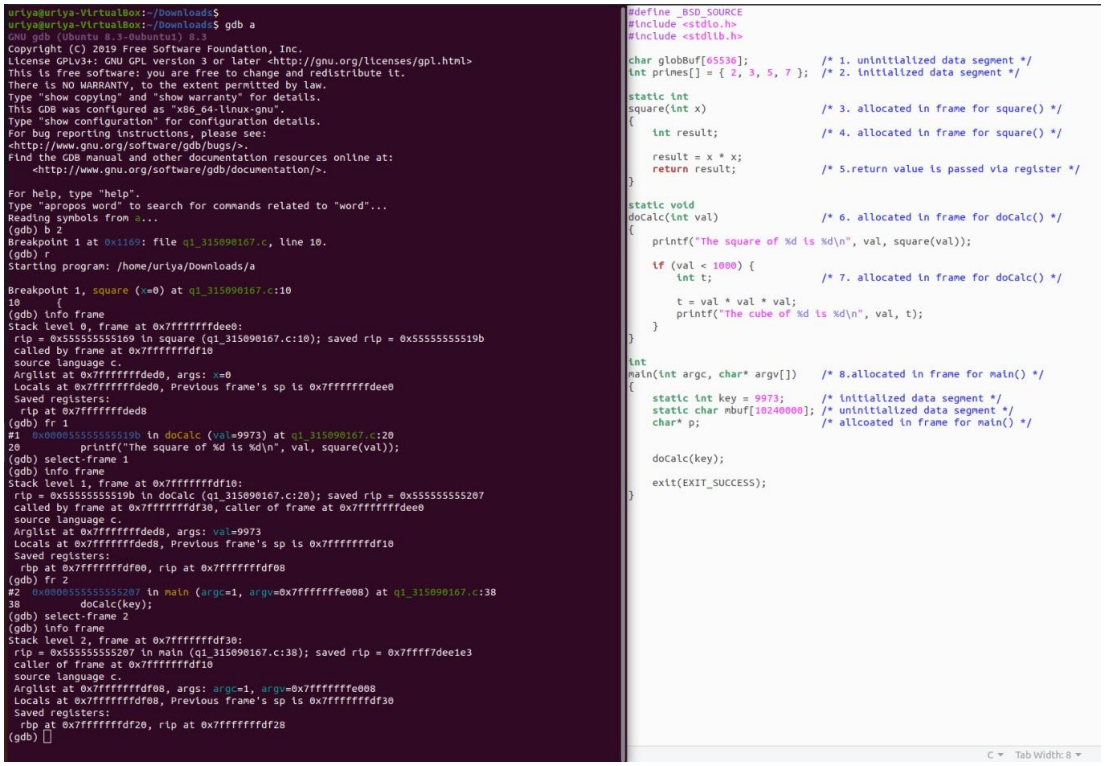


כפי שניתן לראות יש פה תשובה לשאלות 4 , 7 , 11.(לשאלה 7 int t בתוך הפריים אולם התנאי שורה מעליו לעולם לא מתקיים ולכן לא נכנס , אולם אם ייכנס לתנאי אזי המשתנה יגדר בפריים כמו האחרים).
בנוסף נרצה לדעת לגבי כל פריים מידע לגביו ולכן נשתמש בפקודה info frame.
Info frame ייתן לנו את המידה הבא אודות כל frame אותו נבחר:

This command prints a verbose description of the selected stack frame, including:

- the address of the frame
- the address of the next frame down (called by this frame)
- the address of the next frame up (caller of this frame)
- the language in which the source code corresponding to this frame is written
- the address of the frame’s arguments
- the address of the frame’s local variables
- the program counter saved in it (the address of execution in the caller frame)
- which registers were saved in the frame

בס"ד



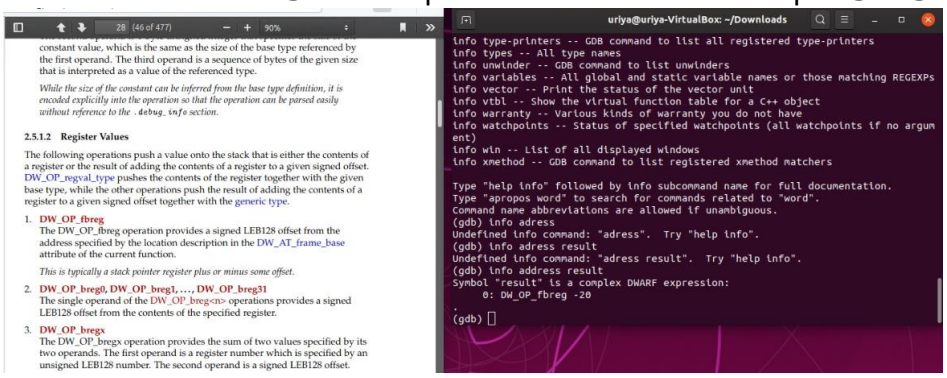
```
uriya@uriya-VirtualBox:~/Downloads$
uriya@uriya-VirtualBox:~/Downloads$ gdb a
GNU gdb (Ubuntu 8.3-0ubuntu1) 8.3
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a...
(gdb) b 2
Breakpoint 1 at 0x1169: file q1_315090167.c, line 10.
(gdb) r
Starting program: /home/uriya/Downloads/a

Breakpoint 1, square (x=0) at q1_315090167.c:10
10      {
(gdb) info frame
Stack level 0, frame at 0x7fffffffdee0:
 rip = 0x555555555169 in square (q1_315090167.c:10); saved rip = 0x55555555519b
 called by frame at 0x7fffffffdf10
 source language c.
 Arglist at 0x7fffffffded0, args: x=0
 Locals at 0x7fffffffded0, Previous frame's sp is 0x7fffffffdee0
 Saved registers:
  rip at 0x7fffffffded8
(gdb) fr 1
#1  0x000055555555519b in doCalc (val=9973) at q1_315090167.c:20
20          printf("The square of %d is %d\n", val, square(val));
(gdb) select-frame 1
(gdb) info frame
Stack level 1, frame at 0x7fffffffdf10:
 rip = 0x55555555519b in doCalc (q1_315090167.c:20); saved rip = 0x555555555207
 called by frame at 0x7fffffffdf30, caller of frame at 0x7fffffffdee0
 source language c.
 Arglist at 0x7fffffffded8, args: val=9973
 Locals at 0x7fffffffded8, Previous frame's sp is 0x7fffffffdf10
 Saved registers:
  rbp at 0x7fffffffdf00, rip at 0x7fffffffdf08
(gdb) fr 2
#2  0x0000555555555207 in main (argc=1, argv=0x7fffffffe008) at q1_315090167.c:38
38          doCalc(key);
(gdb) select-frame 2
(gdb) info frame
Stack level 2, frame at 0x7fffffffdf30:
 rip = 0x555555555207 in main (q1_315090167.c:38); saved rip = 0x7ffff7dee1e3
 caller of frame at 0x7fffffffdf10
 source language c.
 Arglist at 0x7fffffffdf08, args: argc=1, argv=0x7fffffffe008
 Locals at 0x7fffffffdf08, Previous frame's sp is 0x7fffffffdf30
 Saved registers:
  rbp at 0x7fffffffdf20, rip at 0x7fffffffdf28
(gdb)
```

```
#define _BSD_SOURCE
#include <stdio.h>
#include <stdlib.h>

char globBuf[65536];            /* 1. uninitialized data segment */
int primes[] = { 2, 3, 5, 7 };  /* 2. initialized data segment */

static int
square(int x)                   /* 3. allocated in frame for square() */
{
    int result;                 /* 4. allocated in frame for square() */

    result = x * x;
    return result;              /* 5.return value is passed via register */
}

static void
doCalc(int val)                 /* 6. allocated in frame for doCalc() */
{
    printf("The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                  /* 7. allocated in frame for doCalc() */

        t = val * val * val;
        printf("The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char* argv[])    /* 8.allocated in frame for main() */
{
    static int key = 9973;      /* initialized data segment */
    static char mbuf[10240000]; /* uninitialized data segment */
    char* p;                    /* allcoated in frame for main() */


    doCalc(key);

    exit(EXIT_SUCCESS);
}
```

C ▾ Tab Width: 8 ▾

בפי שניתן לראות לגבי הפריים של square נשמר רגיסטר 1 המוחזר – result.(תשובה לשאלה 5).

נשתמש בפקודה info address על result ונקבל את התשובה הבאה:



28 (46 of 477) 90%

constant value, which is the same as the size of the base type referenced by the first operand. The third operand is a sequence of bytes of the given size that is interpreted as a value of the referenced type.

While the size of the constant can be inferred from the base type definition, it is encoded explicitly into the operation so that the operation can be parsed easily without reference to the .debug_info section.

2.5.1.2 Register Values

The following operations push a value onto the stack that is either the contents of a register or the result of adding the contents of a register to a given signed offset. DW_OP_regval_type pushes the contents of the register together with the given base type, while the other operations push the result of adding the contents of a register to a given signed offset together with the generic type.

1. **DW_OP_fbreg**
   The DW_OP_fbreg operation provides a signed LEB128 offset from the address specified by the location description in the DW_AT_frame_base attribute of the current function.

   This is typically a stack pointer register plus or minus some offset.

2. **DW_OP_breg0, DW_OP_breg1, ..., DW_OP_breg31**
   The single operand of the DW_OP_breg<n> operations provides a signed LEB128 offset from the contents of the specified register.

3. **DW_OP_bregx**
   The DW_OP_bregx operation provides the sum of two values specified by its two operands. The first operand is a register number which is specified by an unsigned LEB128 number. The second operand is a signed LEB128 offset.

```
uriya@uriya-VirtualBox: ~/Downloads
info type-printers -- GDB command to list all registered type-printers
info types -- All type names
info unwinder -- GDB command to list unwinders
info variables -- All global and static variable names or those matching REGEXPs
info vector -- Print the status of the vector unit
info vtbl -- Show the virtual function table for a C++ object
info warranty -- Various kinds of warranty you do not have
info watchpoints -- Status of specified watchpoints (all watchpoints if no argum
ent)
info win -- List of all displayed windows
info xmethod -- GDB command to list registered xmethod matchers

Type "help info" followed by info subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) info adress
Undefined info command: "adress".  Try "help info".
(gdb) info adress result
Undefined info command: "adress result".  Try "help info".
(gdb) info address result
Symbol "result" is a complex DWARF expression:
     0: DW_OP_fbreg -20
.
(gdb)
```

הובחה לכך ש result מועבר דרך רגיסטר.
באותו אופן ניתן לדעת לגבי שאר המשתנים.