

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

```

uriya@uriya-VirtualBox: ~/Downloads
uriya@uriya-VirtualBox:~/Downloads$ nm 315090167
000000000004024 B __bss_start
000000000004040 b completed.8055
w __cxa_finalize@GLIBC_2.2.5
000000000004000 D __data_start
000000000004000 W data_start
0000000000010b0 t deregister_tm_clones
000000000001182 t doCalc
000000000001120 t __do_global_dtors_aux
000000000003db8 d __do_global_dtors_aux_fini_array_entry
000000000004008 D dso_handle
000000000003dc0 d _DYNAMIC
000000000004024 D _edata
000000000009d8060 B _end
U _exit@GLIBC_2.2.5
000000000001298 T _fini
000000000001160 t frame_dummy
000000000003db0 d __frame_dummy_init_array_entry
0000000000021cc r __FRAME_END__
000000000003fb0 d __GLOBAL_OFFSET_TABLE__
000000000009c8060 B globBuf
w __gmon_start__
000000000002034 r __GNU_EH_FRAME_HDR
000000000001000 t __init
000000000003db8 d __init_array_end
000000000003db0 d __init_array_start
000000000002000 R __IO_stdin_used
w __ITM_deregisterTMCloneTable
w __ITM_registerTMCloneTable
000000000004020 d key.2841
000000000001290 T __libc_csu_fini
000000000001220 T __libc_csu_init
U __libc_start_main@GLIBC_2.2.5
0000000000011e7 T main
000000000004060 b mbuf.2842
000000000004010 D primes
U printf@GLIBC_2.2.5
0000000000010e0 t register_tm_clones
000000000001169 t square
000000000001080 T _start
000000000004028 D __TMC_END__
uriya@uriya-VirtualBox:~/Downloads$

```

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, **דגש**: 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 **key** ו-**primes** מופיעות האותיות **'d'\D'** המציינות שהסמל נמצא ב **initialized data**. תשובה לשאלות 2 ו-9.
- בסמלים **square**, **doCalc**, **main** 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:

```

Activities Terminal Jul 19 12:48
uriya@uriya-VirtualBox: ~/Downloads
uriya@uriya-VirtualBox:~/Downloads$ objdump -d 315090167

315090167:      file format elf64-x86-64

Disassembly of section .init:

0000000000001000 <_init>:
1000:      f3 0f 1e fa      endbr64
1004:      48 83 ec 08      sub    $0x8,%rsp
1008:      48 8b 05 d9 2f 00 00 mov    0x2fd9(%rip),%rax      # 3fe8 <__gmon_start__>
100f:      48 85 c0          test   %rax,%rax
1012:      74 02            je     1016 <_init+0x16>
1014:      ff d0            callq  *%rax
1016:      48 83 c4 08      add    $0x8,%rsp
101a:      c3              retq

Disassembly of section .plt:

0000000000001020 <_plt>:
1020:      ff 35 92 2f 00 00 pushq  0x2f92(%rip)      # 3fb8 <_GLOBAL_OFFSET_TABLE+0x8>
1026:      f2 ff 25 93 2f 00 00 bnd jmpq *0x2f93(%rip)      # 3fc0 <_GLOBAL_OFFSET_TABLE+0x10>
102d:      0f 1f 00          nopl   (%rax)
1030:      f3 0f 1e fa      endbr64
1034:      68 00 00 00 00 00 pushq  $0x0
1039:      f2 e9 e1 ff ff ff bnd jmpq 1020 <_plt>
103f:      90              nop
1040:      f3 0f 1e fa      endbr64
1044:      68 01 00 00 00 00 pushq  $0x1
1049:      f2 e9 d1 ff ff ff bnd jmpq 1020 <_plt>
104f:      90              nop

Disassembly of section .plt.got:

0000000000001050 <_cxa_finalize@plt>:
1050:      f3 0f 1e fa      endbr64
1054:      f2 ff 25 9d 2f 00 00 bnd jmpq *0x2f9d(%rip)      # 3ff8 <_cxa_finalize@GLIBC_2.2.5>
105b:      0f 1f 44 00 00 00 nopl   0x0(%rax,%rax,1)

110e:      74 08            je     1118 <register_tm_clones+0x38>
1110:      ff e0            jmpq   *%rax
1112:      66 0f 1f 44 00 00 nopw   0x0(%rax,%rax,1)
1118:      c3              retq
1119:      0f 1f 80 00 00 00 nopl   0x0(%rax)

0000000000001120 <__do_global_dtors_aux>:
1120:      f3 0f 1e fa      endbr64
1124:      80 3d 15 2f 00 00 00 cmpb   $0x0,0x2f15(%rip)      # 4040 <completed.8055>
112b:      75 2b            jne    1158 <__do_global_dtors_aux+0x38>
112d:      55              push   %rbp
112e:      48 83 3d c2 2e 00 00 cmpq   $0x0,0x2ec2(%rip)      # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
1135:      00              mov     %rsp,%rbp
1136:      48 89 e5          mov     %rsp,%rbp
1139:      74 0c            je     1147 <__do_global_dtors_aux+0x27>
113b:      48 8b 3d c6 2e 00 00 mov     0x2ec6(%rip),%rdi      # 4008 <__dso_handle>
1142:      e8 09 ff ff ff   callq  1050 <__cxa_finalize@plt>
1147:      e8 64 ff ff ff   callq  10b0 <deregister_tm_clones>
114c:      c6 05 ed 2e 00 00 01 movb   $0x1,0x2eed(%rip)      # 4040 <completed.8055>
1153:      5d              pop     %rbp
1154:      c3              retq
1155:      0f 1f 00          nopl   (%rax)
1158:      c3              retq
1159:      0f 1f 80 00 00 00 nopl   0x0(%rax)

0000000000001160 <frame_dummy>:
1160:      f3 0f 1e fa      endbr64
1164:      e9 77 ff ff ff   jmpq   10e0 <register_tm_clones>

0000000000001169 <square>:
1169:      f3 0f 1e fa      endbr64
116d:      55              push   %rbp
116e:      48 89 e5          mov     %rsp,%rbp
1171:      89 7d ec          mov     %edi,-0x14(%rbp)
1174:      8b 45 ec          mov     -0x14(%rbp),%eax
1177:      0f af c0          imul    %eax,%eax
117a:      89 45 fc          mov     %eax,-0x4(%rbp)
117d:      8b 45 fc          mov     -0x4(%rbp),%eax
1180:      5d              pop     %rbp

```

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

** 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, שאלם 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).

```

1180: 5d          pop     %rbp
1181: c3          retq

0000000000001182: <doCalc>:
1182: f3 0f 1e fa  endbr64
1186: 55          push    %rbp
1187: 48 89 e5     mov     %rsp,%rbp
118a: 48 83 ec 20   sub     $0x20,%rsp
118e: 89 7d ec     mov     %edi,-0x14(%rbp)
1191: 8b 45 ec     mov     -0x14(%rbp),%eax
1194: 89 c7        mov     %eax,%edi
1196: e8 ce ff ff ff callq   1169 <square>
119b: 89 c2        mov     %eax,%edx
119d: 8b 45 ec     mov     -0x14(%rbp),%eax
11a0: 89 c6        mov     %eax,%esi
11a2: 48 8d 3d 5b 0e 00 00 lea     0xe5b(%rip),%rdi # 2004 <_IO_stdin_used+0x4>
11a9: b8 00 00 00 00 mov     $0x0,%eax
11ae: e8 ad fe ff ff callq   1060 <printf@plt>
11b3: 81 7d ec e7 03 00 00 cmpl    $0x3e7,-0x14(%rbp)
11ba: 7f 28        jg      11e4 <doCalc+0x62>
11bc: 8b 45 ec     mov     -0x14(%rbp),%eax
11bf: 0f af c0     imul    %eax,%eax
11c2: 8b 55 ec     mov     -0x14(%rbp),%edx
11c5: 0f af c2     imul    %edx,%eax
11c8: 89 45 fc     mov     %eax,-0x4(%rbp)
11cb: 8b 55 fc     mov     -0x4(%rbp),%edx
11ce: 8b 45 ec     mov     -0x14(%rbp),%eax
11d1: 89 c6        mov     %eax,%esi
11d3: 48 8d 3d 42 0e 00 00 lea     0xe42(%rip),%rdi # 201c <_IO_stdin_used+0x1c>
11da: b8 00 00 00 00 mov     $0x0,%eax
11df: e8 7c fe ff ff callq   1060 <printf@plt>
11e4: 90          nop
11e5: c9          leaveq  %eax
11e6: c3          retq

00000000000011e7: <main>:
11e7: f3 0f 1e fa  endbr64
11eb: 55          push    %rbp
11ee: 48 89 e5     mov     %rsp,%rbp

```

****** ניתן לפתור שאלה זו גם באמצעות שימוש בפקודה size, כאשר כל פעם נוסף עוד שורת קוד מהקוד המקורי שקיבלנו ונריך 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 מימד הקוד שלנו – כמה פריימים יש לנו בקוד.

```

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 0x1109: 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) bt
#0 square (x=0) at q1_315090167.c:10
#1 0x0000555555555190 in doCalc (v=1+9973) at q1_315090167.c:20
#2 0x00005555555551207 in main (argc=1, argv=0x7fffffffe008) at q1_315090167.c:38

```

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.

```

uriya@uriya-VirtualBox: ~/Downloads
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 (<=0) at q1_315090167.c:10
10 {
(gdb) bt
#0  square (<=0) at q1_315090167.c:10
#1  0x0000555555555190 in doCalc (val=9973) at q1_315090167.c:20
#2  0x0000555555555207 in main (argc=1, argv=0x7fffffffe008) at q1_315090167.c:38
(gdb) p 0
#0  square (<=0) at q1_315090167.c:10
(gdb) info local
result = 32767
(gdb) fr 1
#1  0x0000555555555190 in doCalc (val=9973) at q1_315090167.c:20
#2  0x0000555555555207 in main (argc=1, argv=0x7fffffffe008) at q1_315090167.c:38
(gdb) info local
No locals.
(gdb) fr 2
#2  0x0000555555555207 in main (argc=1, argv=0x7fffffffe008) at q1_315090167.c:38
#3  doCalc(key);
(gdb) info local
key = 9973
mbuf = <error reading variable mbuf (value requires 10240000 bytes, which is more than max-value-size)>
p = <optimized out>
(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; /* 5. return value is passed via register */
    return result;
}

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

    if (val < 10000) {
        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[]) /* allocated in frame for main() */
{
    static int key = 9973; /* initialized data segment */
    static char mbuf[10240000]; /* uninitialized data segment */
    char* p; /* allocated in frame for main() */

    doCalc(key);

    exit(EXIT_SUCCESS);
}

```

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


```

url@url:~/Downloads$ gdb a
url@url:~/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 0x1109: file q1_315090167.c, line 10.
(gdb) r
Starting program: /home/url/Downloads/a

Breakpoint 1, square (x=0) at q1_315090167.c:10
10      {
(gdb) info frame
Stack level 0, frame at 0x7ffffffdeeb:
 rbp = 0x55555555109 in square (q1_315090167.c:10); saved rbp = 0x5555555510b
 called by frame at 0x7ffffffdf10
 source language c.
 Argvlist at 0x7ffffffded0, args: x=0
 Locals at 0x7ffffffded0, Previous frame's sp is 0x7ffffffdeeb
 Saved registers:
  rbp at 0x7ffffffded8
(gdb) fr 1
#1 0x00005555555510b 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 0x7ffffffdf10:
 rbp = 0x5555555510b in doCalc (q1_315090167.c:20); saved rbp = 0x555555555207
 called by frame at 0x7ffffffdf30, caller of frame at 0x7ffffffdeeb
 source language c.
 Argvlist at 0x7ffffffded8, args: val=9973
 Locals at 0x7ffffffdf00, Previous frame's sp is 0x7ffffffdf10
 Saved registers:
  rbp at 0x7ffffffdf08, rbp at 0x7ffffffdf08
(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 0x7ffffffdf30:
 rbp = 0x555555555207 in main (q1_315090167.c:38); saved rbp = 0x7ffff7deee3
 caller of frame at 0x7ffffffdf10
 source language c.
 Argvlist at 0x7ffffffdf08, args: argc=1, argv=0x7fffffffe008
 Locals at 0x7ffffffdf00, Previous frame's sp is 0x7ffffffdf30
 Saved registers:
  rbp at 0x7ffffffdf20, rbp at 0x7ffffffdf20
(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; /* 5. return value is passed via register */
    return result;
}

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) { /* 7. allocated in frame for doCalc() */
        int t;

        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; /* 1. initialized data segment */
    static char mbuf[1024000]; /* 2. uninitialized data segment */
    char* p; /* 3. allocated in frame for main() */

    doCalc(key);

    exit(EXIT_SUCCESS);
}

```

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

נשתמש בפקודה info address על result ונקבל את התשובה הבאה:

```

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.

```

```

url@url:~/Downloads$ gdb
url@url:~/Downloads$ gdb
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) info type-printers
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 address
Undefined info command: "address". Try "help info".
(gdb) info address result
Undefined info command: "address result". Try "help info".
(gdb) info address result
Symbol "result" is a complex DWARF expression:
0: DW_OP_fbreg -20
(gdb)

```

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