מגיש: שילה גילאור תז: 302537394

שאלה 1 (25 נק'): מבנה הזכרון של תוכנית בשפת C.

בשאלה זו תתנסו במבנה הזכרון של תוכנית בשפת C הכולל את הקטעים text, stack ,heap וכו'. נתונה התוכנית בשאלה זו באפשרותכם להשתמש (comments). למטרה זו באפשרותכם להשתמש בהתוכנית objdump, nm, size (למדו איך להשתמש בהם).

מה עליכם לבצע:

- 0) למדו איך להשתמש ב- **.objdump**, **nm**, **size** השתמשו ב- **man** או מקורות אחרים לפי בחירתכם. עליכם להכיר את הכלים האלה לפחות ברמה שתאפשר לכם לבצע את המטלה.
 - 1) החליפו כל הערה (comment) שיש בה שאלה בתשובה, בתשובה של שורה אחת בתוך <u>התכנית</u> <u>המקורית</u>.כל השאלות ממוספרות. שמרו על אותו מספור בתשובות שלכם.
- 2) יש ליצור קובץ pdf נפרד, שבו תסבירו כל אחת מתשובותיכם. כמו כן הוסיפו פלט של הכלים(הנ"ל) שהשתמשתם בהם, שמאשר את התשובה שלכם. יש להשתמש באותו מספור. לצורך נוחות הבדיקה, אנא העתיקו לפני כל תשובה את השאלה המקורית + שורת הקוד המתאימה מהתוכנית המקורית.

```
#define _BSD_SOURCE
#include <stdio.h>
#include <stdlib.h>
char globBuf[65536];
int primes[] = { 2, 3, 5, 7 }; /* 2. Where is allocated? */
square(int x)
    int result;
    result = x * x;
    return result;
static void
doCalc(int val)
    printf("The square of %d is %d\n", val, square(val));
    if (val < 1000) {
        t = val * val * val;
         printf("The cube of %d is %d\n", val, t);
main(int argc, char* argv[])
    static int key = 9973;
    static char mbuf[10240000]; /*10. Where is allocated? */
char* p; /*11. Where is allocated? */
    doCalc(key);
    exit(EXIT_SUCCESS);
```

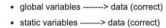
Where I learned from:

https://stackoverflow.com/questions/14588767/where-in-memory-are-my-variables-stored-in-c

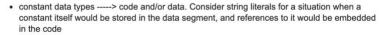


You got some of these right, but whoever wrote the questions tricked you on at least one question:

215









• local variables(declared and defined in functions) -----> stack (correct)

- variables declared and defined in main function ----> heap also stack (the teacher was trying to trick you)
- pointers(ex: char *arr , int *arr) -----> heap data or stack, depending on the context. C lets you declare a global or a static pointer, in which case the pointer itself would end up in
- dynamically allocated space(using malloc, calloc, realloc) -----> stack heap

It is worth mentioning that "stack" is officially called "automatic storage class".

https://linux.die.net/man/1/nm

"B"

"h"

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

"D"

"d"

The symbol is in the initialized data section.

"T"

"†"

The symbol is in the text (code) section.

https://linux.die.net/man/1/objdump

The other common output format, usually seen with ELF based files, looks like this:

```
00000000 1 d .bss 00000000 .bss
00000000 g .text 00000000 fred
```

Here the first number is the symbol's value (sometimes referred to as its address). The next field is actually a set of characters and spaces indicating the flag bits that are set on the symbol. These characters are described below. Next is the section with which the symbol is associated or *ABS* if the section is absolute (ie not connected with any section), or *UND* if the section is referenced in the file being dumped, but not defined there.

After the section name comes another field, a number, which for common symbols is the alignment and for other symbol is the size. Finally the symbol's name is displayed.

The flag characters are divided into 7 groups as follows:

"g"

.. ..

n j n

The symbol is a local (I), global (g), unique global (u), neither global nor local (a space) or both global and local (!). A symbol can be neither local or global for a variety of reasons, e.g., because it is used for debugging, but it is probably an indication of a bug if it is ever both local and global. Unique global symbols are a GNU extension to the standard set of ELF symbol bindings. For such a symbol the dynamic linker will make sure that in the entire process there is just one symbol with this name and type in use.

"O"

The symbol is the name of a function (F) or a file (f) or an object (O) or just a normal symbol (a space).

https://stackoverflow.com/questions/6666805/what-does-each-column-of-objdumps-symbol-table-mean



COLUMN ONE: the symbol's value

COLUMN TWO: a set of characters and spaces indicating the flag bits that are set on the symbol. There are seven groupings which are listed below:



69

group one: (I,g,,!) local, global, neither, both.

0

group two: (w,) weak or strong symbol.

group three: (C,) symbol denotes a constructor or an ordinary symbol.

group four: (W,) symbol is warning or normal symbol.

group five: (I,) indirect reference to another symbol or normal symbol.

group six: (d,D,) debugging symbol, dynamic symbol or normal symbol.

group seven: (F,f,O,) symbol is the name of function, file, object or normal symbol.

COLUMN THREE: the section in which the symbol lives, ABS means not associated with a certain section

COLUMN FOUR: the symbol's size or alignment.

COLUMN FIVE: the symbol's name.

If you want additional information try you man page ;-) or the following links: http://manpages.ubuntu.com/manpages/intrepid/man1/objdump.1.html and http://sourceware.org/binutils/docs/binutils/objdump.html

abana fallani

```
char globBuf[65536];
1) globBuf - Uninitialized data segment – bss - global
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "globBuf"
00000000009c8060 B
shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "globBuf"
00000000009c8060 g
                     0 .bss
                             0000000000010000
 int primes[] = { 2, 3, 5, 7 }; /* 2. Where is allocated? Initialized data segment. — data — global */
primes - Initialized data segment. – data - global
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "primes"
0000000000004010 D
 shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "primes"
                     O .data 00000000000000010
0000000000004010 g
square() - Allocated in frame for square() – text - local
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "square
000000000001169 t
0000000000001169 l
    int result;
result – stack
   return result;
5) return value - Return value passed via register
6) doCalc() - Allocated in frame for doCalc() - text - local
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "doCalc"
000000000001182 t
shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "doCalc"
0000000000001182 l
                     F .text 00000000000<u>0</u>0065
  int t;

 7) t - stack

 main(int argc, char* argv[])
8) main() - Allocated in frame for main() – text - global
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "main"
                U __libc_start_r
                                  @@GLIBC_2.2.5
00000000000011e7 T
shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "main"
                                                          __libc_start_r
                                                                          @@GLIBC_2.2.5
00000000000000000
                     F *UND* 000000000000000
00000000000011e7 g
```

```
static int key = 9973;
9) key - Initialized data segment – data - local
 shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "key"
                     y.2844
0000000000004020 d
 shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "key"
                    .2844
0000000000004020 l
  static char mbuf[10240000]; /*10. Where is allocated? Uninitialized data segment - bss - local */
10) mbuf – Uninitialized data segment – bss - local
shilo@shilo-VirtualBox:~/Desktop/os_final$ nm process_layout_q | grep "mbuf"
0000000000004060 b
                      F.2845
shilo@shilo-VirtualBox:~/Desktop/os_final$ objdump -x process_layout_q | grep "mbuf"
                                                               .2845
0000000000004060 l
                      0 .bss 00000000009<u>c</u>4000
 char* p;
```

11) p - Allocated in frame for main() - Uninitialized