

תרגיל 4 - חלק יבש

המתרגל האחראי על התרגיל: שקד ניסנוב.

כתבו בתיבת **subject**: יבש 4 את"ם.

שאלות בעל-פה ייענו על ידי כל מתרגל.

הוראות הגשה:

- לכל שאלה יש לרשום את התשובה במקום המיועד לכך.
- יש לענות על גבי טופס התרגיל ולהגיש אותו באתר הקורס כקובץ PDF.
- על כל יום איחור או חלק ממנו, שאינו בתיאום עם המתרגל האחראי על התרגיל, יורדו 5 נקודות.
- הגשות באיחור יש לשלוח למייל של אחראי התרגיל בצירוף פרטים מלאים של המגישים (שם+ת.ז).
- שאלות הנוגעות לתרגיל יש לשאול דרך הפיאצה בלבד.
- ההגשה בזוגות.

שאלה 1 (35 נק') – קידוד פקודות:

הסטודנטים וויל וכריס נמצאים כבר שנים רבות ביריבות עיקשת עקב בדיחה שסיפר כריס על חברתו של וויל. וויל, עם ניסיון רב במחשבים, החליט לשגע את המחשב של כריס, ובכך להחזיר לו.
(חברה, השאלה הזאת נכתבה באפריל, כשהאירוע הזה היה טרי. אתם תפתרו את זה באזור יוני, ובטח כל העולם שבח מזה. עדיין, תנסו להיכנס לאווירה 😊)

1. הקומפיילר של כריס פתאום הפסיק לתרגם פקודות לשפת מכונה! עזרו לכריס לתרגם את הפקודות הבאות בצורה תקינה מאסמבלי (AT&T syntax) לשפת מכונה.
הערה: יש למלא את הערכים ב-hexadecimal.

0000000000400082 <L1>:

400082: 66 b9 00 02 mov \$512, %cx

0000000000400086 <L2>:

400086: 4c 8d 15 00 00 00 00 lea 0(%rip), %r10

000000000040008d <L3>:

40008d: 48 8b 74 58 05 mov 0x5(%rax,%rbx,2),%rsi

400092: ff 25 34 12 00 00 jmp *0x1234(%rip)

2. מה יהיה ערכו של רגיסטר %r10 בעת הגעת הקוד לכתובת 0x40008d? 0x40008d

המשך התרגיל בעמוד הבא

3. כריס חשב שהרע מאחוריו, אבל לא היה מוכן לכך שהמעבד שלו עליו סמך יפסיק לעבוד. כריס רוצה לעזור למעבד שלו לתרגם את הרצף הבינארי הבא מפקודות מכונה לפקודות באסמבלי.

67 89 43 42 CC 29 F3 C1 EB 05

הרצף הנ"ל נתון ב-hexadecimal, משמאל לימין (ה-byte הראשון ברצף הוא 0x67). את רצף הפקודות שמקודד ברצף הבינארי עליכם לכתוב בשורות הבאות:

Mov %eax, 0x42(%ebx)

int3

sub %esi, %ebx

shr \$0x5, %ebx

הערות: כל פקודה חייבת להופיע בשורה נפרדת. ניתן להשאיר שורות ריקות.

המשך התרגיל בעמוד הבא

4. וויל החליט שהוא הולך על כל הקופה, והוא חייב שכריס בכלל לא יוכל להגיד את השם של חברתו. לכן, מנע ממנו להשתמש בשם של חברתו ובנוסף, גם בקידודים של הפקודות שנמצאות למטה (כדי ללכת על בטוח). כתוצאה מכך, כריס חייב לקחת פקודות פשוטות ולהאריך\לקצר את קידודן מבלי לפגוע בנכונותן (כלומר, למצוא קידוד שיבצע את מה שהפקודה המקורית מבצעת, אך ארוך\קצר יותר מבחינת כמות bytes). עזרו לכריס לבצע זאת בעזרת השלמת הטבלה הבאה.

הערות:

1. השורה הראשונה הושלמה עבורכם כדוגמה
2. הניחו כי הפקודה בכל שורה מתחילה בכתובת 0x1000.
3. את הקידודים יש לכתוב בבסיס hexadecimal, כאשר ה-byte הראשון הוא השמאלי ביותר.
4. את הפקודה באסמבלי יש לכתוב ב-AT&T syntax
5. **אסור להשתמש ב-SIB או REX אם הפקודה המקורית לא עשתה זאת**
6. **יש להשתמש באותה פקודה בקידוד הפקודה החדשה** (אפשר opcode שונה של אותה פקודה).

<u>קידוד הפקודה המקורית</u>	<u>הפקודה המקורית באסמבלי</u>	<u>קידוד הפקודה הארוכה</u>
EB 50	jmp 0x1052	E9 50 00 00 00
D1 E8	<u>shr \$1, %eax</u>	<u>C1 E8 01</u>
67 8B 03	<u>mov (%ebx), %eax</u>	<u>67 8b 83 00 00 00 00</u>

<u>קידוד הפקודה המקורית</u>	<u>הפקודה המקורית באסמבלי</u>	<u>קידוד הפקודה הקצרה</u>
68 00 00 00 00	-----	-----
E9 FB FF FF FF	<u>Jmp \$-5</u>	<u>EB FB</u>

שאלה 2 (40 נק') – קבצי ELF וקישור סטטי:

גויסתם לצוות cyber security של חברת הייטק גדולה. בבר ביום הראשון הגיע ראש הצוות ושאל אתכם את השאלה הבאה – "האם יעלה על הדעת שסטודנט בטכניון לא ידע לנתח קבצי ELF?". לכן, נתן לכם את המשימה הבאה והשאיר לכם כסף בתן-ביס. לפניהם שלושה קבצים:

file1.asm

```
.global _start, get_password_function

.section .rodata
success_message: .ascii "You have been hacked!\n"
.data
get_password_function: .zero 8

.text
_start:
    callq get_function
    callq *get_password_function
    callq print_success

    movq $60, %rax
    movq (secret_password), %rdi
    syscall

my_strlen:
    push %rbp
    mov %rsp, %rbp

    mov $success_message, %rbx
    xor %rcx, %rcx
check:
    mov (%rbx), %cl
    cmp $0, %cl
    je end

    inc %rbx
    jmp check

end:
    sub $success_message, %rbx
    mov %rbx, %rax
    pop %rbp
    ret

print_success:
    push %rbp
    mov %rsp, %rbp

    call my_strlen
    mov %rax, %rdx
    mov $1, %rax
    mov $1, %rdi
    mov $success_message, %rsi
    syscall

    pop %rbp
    ret
```

file2.asm

```
.global get_function

.text
get_function:
    movq $get_password, get_password_function
    ret
```

file3.c

```
#include <stdio.h>
#include <string.h>

int secret_password = 0;

static void hack() {
    // Function is hacking very hard...
    // Go easy on it...
    // It's his first time...
    // ... ..
    // DONE!
    secret_password = 118;
}

void get_password() {
    hack();
}
```

הוחלט לייצר קובץ ריצה. לכן הורצו הפקודות הבאות:

```
as file1.asm -o file1.o
as file2.asm -o file2.o
gcc file3.c -c -o file3.o
ld file1.o file2.o file3.o -o will_it_run.out
```

1. עבור כל אחד מקבצי ה-`o` שנוצרו לעיל, השלימו את טבלת הסמלים שלהם.

הערות:

1. ניתן להשאיר שורות ריקות.
2. בעמודה `Ndx` עליכם לכתוב את שם ה-`section`, או `UND`.

file1.o Symbol Table:

בתשובתכם אינכם צריכים להתייחס ל-`check` ו-`end` של labels

שם	נראות (Bind)	Ndx (Section)
<code>_start</code>	global	text
<code>get_password_function</code>	global	data
<code>success_message</code>	local	rodata
<code>secret_password</code>	global	Und
<code>print_success</code>	local	text
<code>my_strlen</code>	local	text
<code>get_function</code>	global	Und

file2.o Symbol Table:

שם	נראות (Bind)	Ndx (Section)
<code>get_function</code>	global	text
<code>get_password_function</code>	global	Und
<code>get_password</code>	global	Und

file3.o Symbol Table:

שם	נראות (Bind)	Ndx (Section)
<code>secret_password</code>	global	bss
<code>get_password</code>	global	text
<code>hack</code>	local	text

להלן הפלט של הרצת הפקודה : readelf -S file3.o

There are 12 section headers, starting at offset 0x2c8:

Section Headers:

[Nr]	Name	Type	Address	Offset
	Size	EntSize	Flags Link Info Align	
[0]		NULL	0000000000000000	00000000
	0000000000000000	0000000000000000	0 0 0	
[1]	.text	PROGBITS	0000000000000000	00000040
	0000000000000022	0000000000000000	AX 0 0 1	
[2]	.rela.text	RELA	0000000000000000	00000220
	0000000000000018	0000000000000018	I 9 1 8	
[3]	.data	PROGBITS	0000000000000000	00000062
	0000000000000000	0000000000000000	WA 0 0 1	
[4]	.bss	NOBITS	0000000000000000	00000064
	0000000000000004	0000000000000000	WA 0 0 4	
[5]	.comment	PROGBITS	0000000000000000	00000064
	000000000000002c	0000000000000001	MS 0 0 1	
[6]	.note.GNU-stack	PROGBITS	0000000000000000	00000090
	0000000000000000	0000000000000000	0 0 1	
[7]	.eh_frame	PROGBITS	0000000000000000	00000090
	0000000000000058	0000000000000000	A 0 0 8	
[8]	.rela.eh_frame	RELA	0000000000000000	00000238
	0000000000000030	0000000000000018	I 9 7 8	
[9]	.symtab	SYMTAB	0000000000000000	000000e8
	0000000000000108	0000000000000018	10 9 8	
[10]	.strtab	STRTAB	0000000000000000	000001f0
	000000000000002b	0000000000000000	0 0 1	
[11]	.shstrtab	STRTAB	0000000000000000	00000268
	0000000000000059	0000000000000000	0 0 1	

2. כעת נסתכל על תוכן הקובץ, בעזרת הפקודה file3.o -C hexdump. סמנו על גבי הפלט של hexdump:

a. את טבלת הסמלים של file3 בירוק

b. את ה-strtab באדום

address	data	address	data
00000000	7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00	00000300	00 00 00 00 00 00 00 00 20 00 00 00 01 00 00 00
00000010	01 00 3e 00 01 00 00 00 00 00 00 00 00 00 00 00	00000310	06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020	00 00 00 00 00 00 00 00 c8 02 00 00 00 00 00 00	00000320	40 00 00 00 00 00 00 00 22 00 00 00 00 00 00 00
00000030	00 00 00 00 40 00 00 00 00 00 40 00 0c 00 0b 00	00000330	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00000040	55 48 89 e5 c7 05 00 00 00 00 86 92 03 00 90 5d	00000340	00 00 00 00 00 00 00 00 1b 00 00 00 04 00 00 00
00000050	c3 55 48 89 e5 b8 00 00 00 00 e8 e1 ff ff 90 90	00000350	40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060	5d c3 00 00 00 47 43 43 3a 20 28 55 62 75 6e 74	00000360	20 02 00 00 00 00 00 00 18 00 00 00 00 00 00 00
00000070	75 20 37 2e 34 2e 30 2d 31 75 62 75 6e 74 75 31	00000370	09 00 00 00 01 00 00 00 08 00 00 00 00 00 00 00
00000080	7e 31 38 2e 30 34 2e 31 29 20 37 2e 34 2e 30 00	00000380	18 00 00 00 00 00 00 00 26 00 00 00 01 00 00 00
00000090	14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01	00000390	03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000a0	1b 0c 07 08 90 01 00 00 1c 00 00 00 1c 00 00 00	000003a0	62 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000b0	00 00 00 00 11 00 00 00 00 41 0e 10 86 02 43 0d	000003b0	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
000000c0	06 4c 0c 07 08 00 00 00 1c 00 00 00 3c 00 00 00	000003c0	00 00 00 00 00 00 00 00 2c 00 00 00 08 00 00 00
000000d0	00 00 00 00 11 00 00 00 00 41 0e 10 86 02 43 0d	000003d0	03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000e0	06 4c 0c 07 08 00 00 00 00 00 00 00 00 00 00 00	000003e0	64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	000003f0	00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
00000100	01 00 00 00 04 00 f1 ff 00 00 00 00 00 00 00 00	00000400	00 00 00 00 00 00 00 00 31 00 00 00 01 00 00 00
00000110	00 00 00 00 00 00 00 00 00 00 00 00 01 00 01 00	00000410	30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00000420	64 00 00 00 00 00 00 00 2c 00 00 00 00 00 00 00
00000130	00 00 00 00 03 00 03 00 00 00 00 00 00 00 00 00	00000430	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00000140	00 00 00 00 00 00 00 00 00 00 00 00 03 00 04 00	00000440	01 00 00 00 00 00 00 00 3a 00 00 00 01 00 00 00
00000150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00000450	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160	09 00 00 00 02 00 01 00 00 00 00 00 00 00 00 00	00000460	90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170	11 00 00 00 00 00 00 00 00 00 00 00 03 00 06 00	00000470	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00000180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00000480	00 00 00 00 00 00 00 00 4f 00 00 00 01 00 00 00
00000190	00 00 00 00 03 00 07 00 00 00 00 00 00 00 00 00	00000490	02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001a0	00 00 00 00 00 00 00 00 00 00 00 00 03 00 05 00	000004a0	90 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
000001b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	000004b0	00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00
000001c0	0e 00 00 00 11 00 04 00 00 00 00 00 00 00 00 00	000004c0	00 00 00 00 00 00 00 00 4a 00 00 00 04 00 00 00
000001d0	04 00 00 00 00 00 00 00 1e 00 00 00 12 00 01 00	000004d0	40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001e0	11 00 00 00 00 00 00 00 11 00 00 00 00 00 00 00	000004e0	38 02 00 00 00 00 00 00 30 00 00 00 00 00 00 00
000001f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	000004f0	09 00 00 00 07 00 00 00 08 00 00 00 00 00 00 00
00000200	03 72 05 74 5f 70 01 73 73 73 0f 72 00 00 00 00	00000500	18 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00
00000210	22 3f 70 01 73 73 72 0f 72 00 00 00 00 00 00 00	00000510	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000220	06 00 00 00 00 00 00 00 02 00 00 00 09 00 00 00	00000520	e8 00 00 00 00 00 00 00 08 01 00 00 00 00 00 00
00000230	f8 ff ff ff ff ff ff ff 20 00 00 00 00 00 00 00	00000530	0a 00 00 00 09 00 00 00 08 00 00 00 00 00 00 00
00000240	02 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00	00000540	18 00 00 00 00 00 00 00 09 00 00 00 03 00 00 00
00000250	40 00 00 00 00 00 00 00 02 00 00 00 02 00 00 00	00000550	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000260	11 00 00 00 00 00 00 00 00 2e 73 79 6d 74 61 62	00000560	f0 01 00 00 00 00 00 00 2b 00 00 00 00 00 00 00
00000270	00 2e 73 74 72 74 61 62 00 2e 73 68 73 74 72 74	00000570	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00000280	61 62 00 2e 72 65 6c 61 2e 74 65 78 74 00 2e 64	00000580	00 00 00 00 00 00 00 00 11 00 00 00 03 00 00 00
00000290	61 74 61 00 2e 62 73 73 00 2e 63 6f 6d 6d 65 6e	00000590	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002a0	74 00 2e 6e 6f 74 65 2e 47 4e 55 2d 73 74 61 63	000005a0	68 02 00 00 00 00 00 00 59 00 00 00 00 00 00 00
000002b0	6b 00 2e 72 65 6c 61 2e 65 68 5f 66 72 61 6d 65	000005b0	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
000002c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

כעת נביט בקוד המכונה שנוצר עבור כל אחד מ3 הקבצים, באמצעות פקודת objdump:

file1.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <_start>:

0:	e8 00 00 00 00	callq 5 <_start+0x5>
5:	ff 14 25 00 00 00 00	callq *0x0
c:	e8 37 00 00 00	callq 48 <print_success>
11:	48 c7 c0 3c 00 00 00	mov \$0x3c,%rax
18:	48 8b 3c 25 00 00 00	mov 0x0,%rdi
1f:	00	
20:	0f 05	syscall

0000000000000022 <my_strlen>:

22:	55	push %rbp
23:	48 89 e5	mov %rsp,%rbp
26:	48 c7 c3 00 00 00 00	mov \$0x0,%rbx
2d:	48 31 c9	xor %rcx,%rcx


```

0000000000000030 <check>:
 30: 8a 0b                mov     (%rbx),%cl
 32: 80 f9 00             cmp     $0x0,%cl
 35: 74 05                je      3c <end>
 37: 48 ff c3             inc     %rbx
 3a: eb f4                jmp     30 <check>

000000000000003c <end>:
 3c: 48 81 eb 00 00 00 00 sub     $0x0,%rbx
 43: 48 89 d8             mov     %rbx,%rax
 46: 5d                   pop     %rbp
 47: c3                   retq

0000000000000048 <print_success>:
 48: 55                   push    %rbp
 49: 48 89 e5             mov     %rsp,%rbp
 4c: e8 d1 ff ff ff      callq   22 <my_strlen>
 51: 48 89 c2             mov     %rax,%rdx
 54: 48 c7 c0 01 00 00 00 mov     $0x1,%rax
 5b: 48 c7 c7 01 00 00 00 mov     $0x1,%rdi
 62: 48 c7 c6 00 00 00 00 mov     $0x0,%rsi
 69: 0f 05               syscall
 6b: 5d                   pop     %rbp
 6c: c3                   retq

```

file2.o: file format elf64-x86-64

Disassembly of section .text:

```

0000000000000000 <get_function>:
 0: 48 c7 04 25 00 00 00 movq    $0x0,0x0
 7: 00 00 00 00 00 00
 c: c3                   retq

```

file3.o: file format elf64-x86-64

Disassembly of section .text:

```

0000000000000000 <hack>:
 0: 55                   push    %rbp
 1: 48 89 e5             mov     %rsp,%rbp
 4: c7 05 00 00 00 00 76 movl    $0x76,0x0(%rip)
 b: 00 00 00
 e: 90                   nop
 f: 5d                   pop     %rbp
10: c3                   retq

0000000000000011 <get_password>:
11: 55                   push    %rbp
12: 48 89 e5             mov     %rsp,%rbp
15: b8 00 00 00 00      mov     $0x0,%eax
1a: e8 e1 ff ff ff      callq   0 <hack>
1f: 90                   nop
20: 5d                   pop     %rbp
21: c3                   retq

```

3. לכל קובץ ענו, כמה טבלאות relocation קיימות? לאיזה section שייכת כל טבלה?

a. file1.o – טבלת relocation יחידה עבור text section.

b. file2.o – טבלת relocation יחידה עבור text section.

c. file3.o – טבלת relocation יחידה עבור text section.

הערה: אין צורך לייחס חשיבות לטבלה של eh_frame section (מופיעה ב-file3.o) – היא אינה בחומר הקורס.

4. כעת, עבור כל אחד מקבצי ה-o, השלימו את טבלאות ה-relocation. כל טבלה שאתם כותבים צריכה להכיל את ארבע העמודות הבאות:

file1.o

Offset	Type	Symbol Name	Addend
0x1	יחסי	get_function	-4
0x8	קבוע	.data	0
0x1d	קבוע	secret_passwor	0
0x29	קבוע	.rodata	0
0x3f	קבוע	.rodata	0
0x65	קבוע	.rodata	0

file2.o

Offset	Type	Symbol Name	Addend
0x4	קבוע	get_password	0
0x8	קבוע	get_password_function	0

file3.o

Offset	Type	Symbol Name	Addend
0x06	יחסי	secret_password	-8

כאשר ב-Type ניתן להשלים רק "יחסי" או "קבוע" ואין צורך להשתמש בשמות המלאים.

5. ראש הצוות חזר אחרי ישיבות רבות עם הצוות הקוריאני בנוגע למוצר החדש שהחברה מפתחת. הוא מסביר שהוא חייב אתכם לפרויקט הזה, אבל רק אם אתם מבינים מה התוכנית עושה!

האם בניית התוכנית תצליח (יצירת ה-executable)? **כן** / לא

אם לא, מדוע?

אם כן, מה יודפס למסך ומה יהיה ערך היציאה (exit status) שלה?

יודפס למסך: "You have been hacked!"

ערך יציאה: 118.

6. מה היה קורה אם במקום השורה `callq *get_password_function` בקובץ `file1.asm` היה נכתב `call hack`?

נקבל שגיאת לינקר (קישור) מפני שהפונקציה `hack` מוגדרת כפונקציה סטטית ב-`file3.c` ולכן ה-`bind` שלה יהיה `local` בטבלת הסמלים. בעת קריאה לפונקציה מקובץ `file1.o`, מכיוון שהלינקר "לא רואה" סמלים לוקאליים, הוא לא ימצא את סמל הפונקציה `hack` ואז נקבל שגיאת קישור.

שאלה 3 (25 נק') – קישור דינמי:

שמעון, מרצה באוניברסיטה הפתוחה למנהל טכנולוגי בחולון על שם סמי שמעון, מעוניין לבחון את תלמידיו בקורס עת"מ על החומר שנלמד בסמסטר אביב 2022. לשם כך, הוא כתב קוד שיבחר תשובות רנדומליות לכל שאלה במבחן. שמעון כתב ספריה שתבצע את מילוי התשובות וקוד ראשי שיקרא לה, וידפיס את התשובה לכל שאלה.

הסבר קצר על שורה 7 בקובץ `libhw4.c`: אנחנו מאתחלים את מייצר המספרים הפסאודו רנדומליים (לא באמת רנדומליים) לזמן הנוכחי כדי שנקבל תוצאות שונות בכל הרצה של הקובץ. הסבר יותר מפורט:

If you simply use `rand()` and run your code multiple times you will notice that you tend to get the same sequence of random numbers every time. `srand` is used to seed the random number generator. This allows you to generate different sequences.

main.c

```
1 #include <stdio.h>
2
3 #define NUMBER_OF_QUESTIONS 20
4 void get_answers(char answers[], int n);
5
6 int main() {
7     char answers[NUMBER_OF_QUESTIONS] = {0};
8
9     get_answers(answers, NUMBER_OF_QUESTIONS);
10    for(int i = 0; i < NUMBER_OF_QUESTIONS; i++) {
11        printf("The answer for question %d is %c\n", i + 1, answers[i]);
12    }
13    return 0;
14 }
```

libhw4.c

```
1 #include <time.h>
2 #include <stdlib.h>
3
4 #define NUMBER_OF_OPTIONS 4
5
6 void get_answers(char answers[], int n) {
7     srand(time(NULL));
8     char options[] = {'A', 'B', 'C', 'D'};
9
10    for(int i = 0; i < n; i++) {
11        answers[i] = options[rand() % NUMBER_OF_OPTIONS];
12    }
13 }
```

שמעון בחר לקמפל את הספרייה לקובץ `shared object` ולהשתמש בקשר הדינמי. אלו הפקודות שהריץ:

```
gcc -shared -fPIC -o libhw4.so libhw4.c
```

```
sudo mv libhw4.so /usr/lib/
```

```
gcc -no-pie -o main.out main.c /usr/lib/libhw4.so -Wl,-z,now
```

המשך התרגיל בעמוד הבא

1. האם השימוש בדגל fPIC- ביצירת libhw4.so הכרחי? מדוע?

תשובה:

הדגל הכרחי מפני שבעת יצירת ספרייה דינמית נרצה שהקוד בה יהיה pic וזאת מפני שבעת קריאה

לפונקציה בספרייה מקובץ חיצוני הדרך היחידה של הלינקר למצוא אותה יהיה באמצעות מיקומה

היחסי בקובץ. כמובן שהדרך לקמפל קובץ כך שהוא יהיה position-independent היא באמצעות

הוספת הדגל fPIC- ולכן הוא נדרש.

2. לצערו של שמעון, הוא גילה שרוב התשובות שקיבל היו התשובה הידועה לשמצה C. כדי להקשות על התלמידים שלא באו מוכנים לבחינה, הוא שינה את הקוד והוסיף סיכוי גבוה יותר לקבל אותיות אחרות. מה היתרון בשימוש בקישור דינמי, לעומת הקישור הסטטי, כאשר נדרשים לבצע תיקונים בספרייה?

תשובה:

היתרון הוא שיש לקמפל מחדש רק את הספרייה עצמה ולא את כל הקבצים יחד. ככה בעת הרצת

קובץ הריצה, בגלל שהקישור הדינמי יעבוד כרגיל התוכנית תרוץ כמצופה. לעומת זאת, בקישור

סטטי, בעת שינוי באחד הקבצים יש לקמפל את כולם מחדש יחד על מנת ליצור את קובץ הריצה

החדש.

3. יוסף הריץ את הפקודה objdump -d main.out כדי לחקור את ה-PLT. להלן חלק מהפלט שקיבל:
Disassembly of section .plt:

0000000000400550 <.plt>:

400550:	ff 35 72 0a 20 00	pushq	0x200a72(%rip)
400556:	ff 25 74 0a 20 00	jmpq	*0x200a74(%rip)
40055c:	0f 1f 40 00	nopl	0x0(%rax)

[. . .]

0000000000400570 <printf@plt>:

400570:	ff25 6a 0a 20 00	jmpq	*0x200a6a(%rip)
400576:	68 01 00 00 00	pushq	\$0x1
40057b:	e9 d0 ff ff ff	jmpq	400550 <.plt>

0000000000400580 <get_answers@plt>:

400580:	ff 25 62 0a 20 00	jmpq	*0x200a62(%rip)
400586:	68 02 00 00 00	pushq	\$0x2
40058b:	e9 c0 ff ff ff	jmpq	400550 <.plt>

a. השלימו את קידוד הפקודה של הפקודות בשורות 0x400570-ו 0x400580.

המשך התרגיל בעמוד הבא

b. נתמקד כעת בפקודה בכתובת 0x400580.

i. מהו סוג הקפיצה בו משתמשים? הקפיצה היא אבסולוטית.

ii. מהו סוג האופרנד (אם מדובר בכתובת, ציינו שיטת מיעון)?

שיטת המיעון בא משתמשים היא rip-relative ובאופרנד rip+disp32.

iii. מהי הכתובת אליה נקפוץ בביצוע הקפיצה (אם לא ניתן לדעת, מה כן ידוע עליה)?

לא ניתן לדעת מה תהיה הכתובת אליה נקפוץ מפני שאנו מבצעים את הקישור

הדינמי לא באמצעות lazy binding אלא ישר ממלאים את הכתובות של הפונקציות

במקומות המתאים לפני תחילת הריצה בפועל של הקובץ. מה שכן, הכתובת לא

תשתנה במהלך הריצה – התוכן ב-GOT לא ישתנה ותמיד תהיה שמורה שם

הכתובת אליה נקפוץ בעת קריאה לפונקציה.

c. עזרו לשמעון להשלים את טבלת ה-relocation של הrel.plt (PLT)

בנוסף, נמקו במשפט איך החלטתם על הערך בתא offset.

Offset	Symbol Name	Addend
0x600fe0	printf	0
0x600fe8	get_answers	0

הסבר: הכתובות ב-offset יהיו הכניסות המתאימות לפונקציות ב-GOT - מהן נדע להגיע לכתובת

הפונקציה במהלך ריצת התוכנית. החישוב יתבצע לפי הפקודה שנמצאת בכניסה ב-plt:

$$1. \quad 0x600fe0 = 0x200a6a + 0x400576$$

$$2. \quad 0x600fe8 = 0x200a62 + 0x400586$$

d. בזמן ריצת main.out, מה תכיל הכתובת 0x600fe8 לפני ביצוע שורה מספר 9 בקוד main.c?

תשובה:

לפני תחילת ריצת הקובץ בפועל, ובפרט לפני הקריאה הראשונה ל-get_answers. ה-GOT

יכיל את הכתובת בה שמורה הפונקציה בגלל שהקימפול נעשה עם הדגל now. תוכן

הכתובת לא ישתנה במהלך הריצה.