The Bin Exercise

关于作业：

各位小伙伴，协会的第一次二进制初章已经结束了。希望大家能通过这次培训窥探到宏大的二进制世界，激发探索底层的兴趣。

这是第一次二进制培训的配套练习。本次的作业会需要汇编语言的知识，可以先去学习了解一下。第一题看不太懂可以先看看第二题，毕竟第二题以动手调试为主。

请尽量在 12 月 20 之前写完，做完请直接发给幼稚园（danisjiang@qq.com）**源码编码规范必须采用 UTF-8。附件须为压缩包，压缩包命名格式“Bin1_你的名字”,如“Bin1_张三.zip”。该压缩包须包含每道题的相关文件，命名格式“Bin1_题号名字”，如 第 1 题“Bin1_01_张三.c”。**（请不要在文件名中加空格）

在代码文件中可以注明疑问和想法，我会抽时间将每份代码看过去并回复大家。

可能不是所有的小伙伴都能做出所有的题，没有关系，能够做出没有标明选做的题，你已经很厉害了。只要把会写的写就 OK，不会的空着没事的。但有一点，不要抄，尽量写。可以参考网上的，但不要直接抄网上的，直接抄下来不去思考不去写，没有任何意义。

如有任何关于作业的问题，直接提出来就好。

1. 反汇编

熟练阅读汇编代码是二进制中非常重要的一项技能。现在，我们拿到了一份在 Linux x64 环境下的汇编代码，请试着直接分析它，并提交功能相同的 C 语言代码。

```
1.      .file   "demo.c"
2.      .intel_syntax noprefix
3.      .section    .rodata
4.  .LC0:
5.      .string "%8s"
6.  .LC1:
7.      .string "%d\n"
8.      .text
9.      .globl  main
10.     .type   main, @function
11.main:
12..LFB0:
13.     .cfi_startproc
14.     push    rbp
15.     .cfi_def_cfa_offset 16
16.     .cfi_offset 6, -16
17.     mov rbp, rsp
18.     .cfi_def_cfa_register 6
19.     push    rbx
20.     sub rsp, 56
21.     .cfi_offset 3, -24
22.     mov rax, QWORD PTR fs:40
23.     mov QWORD PTR [rbp-24], rax
24.     xor eax, eax
25.     lea rax, [rbp-48]
26.     mov rsi, rax
27.     mov edi, OFFSET FLAT:.LC0
28.     mov eax, 0
29.     call    __isoc99_scanf
30.     mov DWORD PTR [rbp-56], 0
31.     mov DWORD PTR [rbp-52], 0
32.     jmp .L2
33..L5:
34.     sal DWORD PTR [rbp-56], 4
35.     mov eax, DWORD PTR [rbp-52]
36.     cdqe
```

```
37.     movzx   eax, BYTE PTR [rbp-48+rax]
38.     cmp al, 47
39.     jle .L3
40.     mov eax, DWORD PTR [rbp-52]
41.     cdqe
42.     movzx   eax, BYTE PTR [rbp-48+rax]
43.     cmp al, 57
44.     jg  .L3
45.     mov eax, DWORD PTR [rbp-52]
46.     cdqe
47.     movzx   eax, BYTE PTR [rbp-48+rax]
48.     movsx   eax, al
49.     sub eax, 48
50.     add DWORD PTR [rbp-56], eax
51.     jmp .L4
52. .L3:
53.     mov eax, DWORD PTR [rbp-52]
54.     cdqe
55.     movzx   eax, BYTE PTR [rbp-48+rax]
56.     cmp al, 96
57.     jle .L4
58.     mov eax, DWORD PTR [rbp-52]
59.     cdqe
60.     movzx   eax, BYTE PTR [rbp-48+rax]
61.     cmp al, 102
62.     jg  .L4
63.     mov eax, DWORD PTR [rbp-52]
64.     cdqe
65.     movzx   eax, BYTE PTR [rbp-48+rax]
66.     movsx   eax, al
67.     sub eax, 88
68.     add DWORD PTR [rbp-56], eax
69. .L4:
70.     add DWORD PTR [rbp-52], 1
71. .L2:
72.     mov eax, DWORD PTR [rbp-52]
73.     movsx   rbx, eax
74.     lea rax, [rbp-48]
75.     mov rdi, rax
76.     call    strlen
77.     cmp rbx, rax
78.     jb  .L5
79.     mov eax, DWORD PTR [rbp-56]
```

```
80.     mov esi, eax
81.     mov edi, OFFSET FLAT:.LC1
82.     mov eax, 0
83.     call    printf
84.     mov eax, 0
85.     mov rdx, QWORD PTR [rbp-24]
86.     xor rdx, QWORD PTR fs:40
87.     je  .L7
88.     call    __stack_chk_fail
89. .L7:
90.     add rsp, 56
91.     pop rbx
92.     pop rbp
93.     ret
94. .LFE0:
95.     .size   main, .-main
96.     .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
97.     .section    .note.GNU-stack,"",@progbits
98.
```

2. 实践出真知

在 Linux 环境下，安装 gdb，并安装插件（pwndbg、peda 等），然后用 C 语言写一个简单的程序后编译。试着用 gdb 调试，在调试过程中，请关注一下内容：

1. 内存中各个区段都在哪里（它们每次运行时的地址、大小是一样的吗？），存放着什么内容，有什么特性呢？可以结合搜索引擎一起了解（vmmap 命令可以查看内存分布）
2. 栈，看看栈上都存储着什么东西。以及，函数在执行到 ret 命令时为什么 rsp 刚好指向返回地址？

3. （可选）调用库函数（例如：puts）时，会跳转到哪里？看看程序在真正调用 puts 函数前都做了些什么准备。以及第二次再调用 puts 函数时，过程和第一次调用有什么区别?

关于这道题目，请在文档中写下问题的答案和自己的看法，如果有相关的问题也可以截图提问