实验目的：复习C语言，理解C语言程序的内存结构，包括代码段、数据段的存储内容，特别是栈帧的结构。另外，还有指针的原理和用法。

实验目标：自己设计并编写实验程序，至少要能够实现以下功能：

本程序中，全局变量和局部变量的地址显示，常量、函数的地址显示。

一个函数栈帧结构的显示。

两级指针的工作原理。

自己的扩展目标： 可供参考的内容：

编译生成一个动态连接库，调用其中的函数，展示不同程序调用同一个函数时的地址。

利用调试程序gdb，查看各变量地址

实验原理：

利用C语言中指针的各种灵活特性完成实验。

利用gcc完成动态链接库的编译。

利用gdb实现动态调试。

实验方案：

(1). 输出变量、常量地址较为简单，直接使用&取地址后以%p形式输出即可。输出函数地址有一定难度。函数开始地址与输出变量地址做法一致。由于我用的是64位机器，根据栈帧相关知识，函数返回的地址存在 0x8(%rbp)的位置，因此使用C语言内嵌汇编语句取出其中的值。 ps. 32位机器为0x4(%ebp)，64位为0x8(%rbp)

(2). 由1中获得的函数开始地址与返回地址，获取它们之间的值，即为栈帧内容。

(3).两级指针的第一级为普通指针，第二级为指向指针的指针。

利用gcc编译动态链接库，使用两个.c文件分别调用。

使用gdb查看已经编译好的文件。

实验结论：

运行截图如下。代码见下一页，也可在我的github上查看：

<https://github.com/modricwang/BUAASoft-OS-Labs/tree/master/Lab1>

图片请放大后查看，如果由于压缩导致图片不清晰，也请移步至上述github地址查看，带来的不便敬请谅解！

#include <stdio.h>

void show\_addr(const char \*name, const void \*p) {

printf("%s address = %016p\n", name, p);

}

void get\_register\_val() {

void \*rbp\_val;

\_\_asm\_\_ \_\_volatile\_\_ (

"movq 0x8(%%rbp),%0":"=r"(rbp\_val)

);

show\_addr("Function return", rbp\_val);

void \*begin\_addr = &(get\_register\_val);

show\_addr("Function begin ", begin\_addr);

printf("\n\nStack Frame Begin:\n\n");

for (long long \*ll = rbp\_val; ll >= begin\_addr; ll--) {

printf("%016llx\n", \*ll);

}

printf("\nStack Frame End.\n\n\n");

}

const int p = 10;

int x;

int main() {

int y;

show\_addr("Global var", &x);

show\_addr("Global const", &p);

show\_addr("Local var", &y);

puts("");

get\_register\_val();

long long val = 0xfff;

long long \*l1\_ptr = &val;

long long \*\*l2\_ptr = &l1\_ptr;

if (\*\*l2\_ptr == val) {

printf("Yes, L2 pointer works!\n");

}

}