1. C函数返回struct类型是如何实现的？

A screenshot of a cell phone

Description automatically generated

Slides如上。这里，return\_struct函数返回了一个TEST\_Struct，这是通过function1隐式的传入了一个栈上的地址作为第一个参数（$rdi），那么return\_struct函数只要把返回的struct的内容填入这个地址即可。从代码中也可以看到，return\_struct函数向rdi指向的地址写入了对应的结构题的内容，同时把返回值 $rax 赋值为 $rdi ，正好是结构体所在的内存位置。下面是栈内容的一个描述：

Rdi + 16: eee

Rdi + 12: ddd

Rdi + 8: coo

Rdi + 4: bye

Rdi: age

Rdi – 4: movl $0, %eax 的地址

这只是课件中提供的一种传递方法，在不同的ABI下可能对不同的struct有不同的处理方式，例如在 struct 比较小的时候，可能仍然通过寄存器返回。

2、 C函数是如何传入struct类型参数的？

A screenshot of a cell phone

Description automatically generated

这里 function2 向 input\_struct 传入了一个结构题，具体的方法是，首先在栈上把 TEST\_Struct的各个值填好，然后直接调用 input\_struct 函数。由于参数是 struct ，input\_struct 会认为输入的参数在栈上而不是寄存器中，于是直接从栈上取出对应偏移的数据，当然了要跳过call压入的返回地址。栈的内容（Rsp为 call input\_struct之前的值）：

Rsp + 36: ddd

Rsp + 32: coo

Rsp + 28: bye

Rsp + 24: age

Rsp + 16: eee

Rsp + 12: ddd

Rsp + 8: coo

Rsp + 4: bye

Rsp: age

Rsp - 8: addp $56, %rsp 的地址

可以看到此时的 Rsp 开始就是一个完整的 TEST\_Struct，之上还有一些 function2 自己保存下来的数据，input\_struct 函数也不会去修改它们。

这只是课件中提供的一种传递方法，在不同的ABI下可能对不同的struct有不同的处理方式，例如在struct大小足够小的时候，可能会通过寄存器来传递。

3、有如下C代码（X86-64 Linux系统），编译成对应的汇编代码，请对照两边代码给出M、N的值。同时请在汇编代码段中标出哪些语句的作用是直接将源结构的字段复制给目标结构（每一个字段的复制指令都需分别标出）。

copy\_element:

pushq %rbp

movq %rsp, %rbp

movl %edi, -4(%rbp)

movl %esi, -8(%rbp)

movl -8(%rbp), %eax

movslq %eax, %rcx # rcx为 j

movl -4(%rbp), %eax

movslq %eax, %rdx # rdx 为 i

movq %rdx, %rax # rax = i

addq %rax, %rax # rax = 2 \* i

addq %rdx, %rax # rax = 3 \* i

salq $2, %rax # rax = 12 \* i

addq %rdx, %rax # rax = 13 \* i

addq %rcx, %rax # rax = 13 \* i + j

salq $4, %rax # rax = 16\*(13\* i+j)

leaq mat1(%rax), %rcx # rcx = &mat1[i][j]

movl -4(%rbp), %eax # rax = i

movslq %eax, %rsi # rsi = i

movl -8(%rbp), %eax # rax = j

movslq %eax, %rdx # rdx= j

movq %rdx, %rax # rax = j

salq $3, %rax # rax = 8 \* j

subq %rdx, %rax # rax = 7 \* j

addq %rsi, %rax # rax = 7 \* j + i

salq $4, %rax # rax = 16 \* (7\* j + i)

addq $mat2, %rax # rax = &mat2[j][i]

movq 8(%rax), %rdx # read attribute\_{1,2}

movq (%rax), %rax # read attribute\_3

movq %rax, (%rcx) # write attribute\_3

movq %rdx, 8(%rcx) # write attribute\_{1,2}

movl $0, %eax # return 0

popq %rbp

ret

~

typedef struct typeTAG

{

double attribute\_3;

char attribute\_2;

int attribute\_1;

} TAG;

TAG mat1[M][N];

TAG mat2[N][M];

int copy\_element(int i, int j)

{

mat1[i][j] = mat2[j][i];

return 0;

}

M = 7

N = 13

各个指令的用途已经在右边注明。

4、 设计编写汇编程序，要求如下：

（1）程序命令行输入两个文件名，都是文本文件，名字分别为input.txt与output\_line\_num.txt。要求是统计第一个文件中的文字行数，’LF’（即换行符，ASCII码为0x0A）为分割行的标记，如果最后的文字直到文件末尾也没有LF的话，也算做一行；然后将行数以字符串形式写入第二个文件。

（2）要求在程序内实现一个名为getline的函数，带三个参数，第一个为读取的正文文件的文件描述符，第二个是存放一行文字（包含结尾的0，但不包括LF）的内存地址，第三个是该块内存的可用长度。返回值是读入的文字的长度（不包括0与LF），返回-1表示调用失败，如果已达到文件结尾，可返回0。

要求提供源码，运行截图，以及单独成文的设计说明文字。

代码在目录 wc 下，通过 make 可以生成一个名为 wc 的程序，符合（1）的要求，可以执行 ./wc pg1041.txt lines 对莎士比亚的十四行诗进行行数计数，结果与 wc -l pg1041.txt 一致：

A black sign with white text

Description automatically generated

当文件不存在或者创建失败的时候也会输出错误信息：

A picture containing device, gauge, object

Description automatically generated

或者在argc不对时输出错误信息：

A black sign with white text

Description automatically generated

代码由多个汇编源码文件组成：getline.S putnum.S syscall.S 和 wc.S 其中 getline.S 实现了题目要求的 getline() 函数，稍有不同的点在用使用 -1 表示 EOF ，这样得以区分了空行和EOF的情况。并且在读取到缓冲区长度减一个数的字符时直接返回，不会尝试读下一个字符；Putnum.S实现了非负整数的输出；syscall.S 封装了几个需要用到的 syscall；wc.S为主程序，从栈上读取argc 和 argv ，进行行数计数之后退出。

Makefile：利用自带的规则进行as的调用，最后用ld链接在一起，得到可执行程序 wc

Getline：首先检查输入的参数，然后通过 read 的系统调用逐个字符读取；遇到回车或者缓冲区已满就设置返回值并返回，遇到错误或者EOF就提前退出，错误返回 -2 ，EOF返回 -1 。

Putnum：如果是单个数位，就直接输出；否则进行一次除法，然后递归调用自己，再输出当前数位。只支持非负整数。

Wc：从栈上读取出argc，进行判断，如果不正确就输出错误信息；从栈上继续读取两个参数，注意跳过argv[0]，然后打开输入和输出文件，分别存在寄存器中，也做了错误处理；循环调用getline进行读取，如果返回值介于[0,len-2]之间，说明目前处在一个行的末尾，计数加一，EOF和错误则跳出循环。最后调用 putnum 输出结果到文件中。

代码中多次涉及寄存器的选取和保存，需要注意的是，syscall可能破坏 rcx 和 r11 的值，故尽量不要跨越 syscall 使用这两个寄存器，如果要使用也需要保存；临时变量也尽量按照SysV ABI的callee/caller-saved registers来选择，需要保存的时候临时push/pop，不过也没有特别严格去遵守；由于栈的地址变化都是比较固定的，所以没有在函数开始和结束使用常见的 prologue 和 epilogue。