

C 语言函数传参与返回复杂结构体的机制

石曜铭

2025 年 4 月 29 日

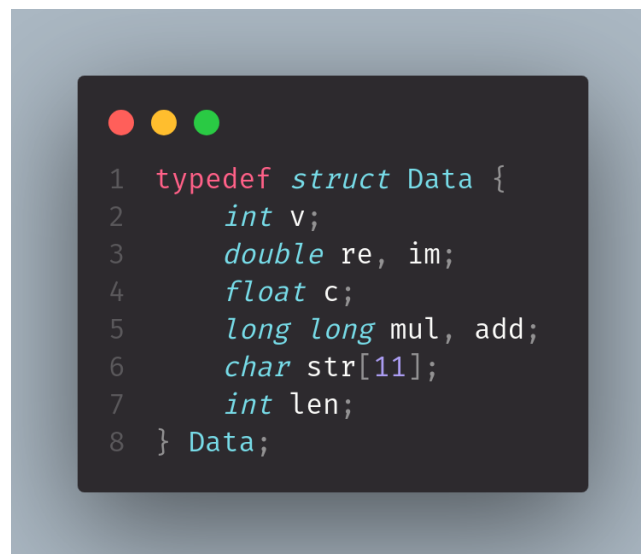
1 实验概述

本次实验中，我使用 C 语言定义了结构体 `Data`，并实现传递和返回 `Data` 类型参数的函数，然后使用编译器编译成汇编代码，通过分析汇编代码中传参和返回值部分的地址偏移量，确定传递复杂类型参数的方式。

此报告中所有汇编语言代码均使用 x86-64 的 AT&T 语法。

2 示例代码

接下来给出实验用到的 C 语言代码。



```
1 typedef struct Data {  
2     int v;  
3     double re, im;  
4     float c;  
5     long long mul, add;  
6     char str[11];  
7     int len;  
8 } Data;
```

图 1: 结构体定义

结构体 `Data` 包括 `int` 类型的 `v`，`double` 类型的 `re, im`，等其他类型，是一个复杂的数据类型。

```

1 Data calc(Data a, Data b) {
2     Data res;
3     res.v = a.v + b.v;
4
5     res.re = a.re * b.re - a.im * b.im;
6     res.im = a.re * b.im + a.im * b.re;
7
8     res.c = (a.c + b.c) / 2;
9
10    res.mul = a.mul * b.mul;
11    res.add = a.add + a.mul * b.add;
12
13    res.len = a.len + b.len;
14    if(res.len > 10) res.len = 10;
15    for(int i = 0; i < res.len; ++i)
16        res.str[i] = (i < a.len) ? a.str[i] : b.str[i - a.len];
17    res.str[res.len] = 0;
18    return res;
19 }

```

图 2: calc 函数定义

函数 calc 计算了 v 的加法，复数 (re, im) 的乘法，浮点数 c 的平均数，先乘后加标签的合并，与字符串的拼接。函数需要用到输入的两个 Data 类型的所有成员，计算互不相同，可以在汇编代码中容易区分出是哪一个成员。

```

1 Data foo(Data a, Data b, Data c) {
2     Data d = calc(a, b);
3     d = calc(c, d);
4     return d;
5 }

```

图 3: 调用 calc 函数示例

Data foo(Data a, Data b, Data c): 调用 calc 两次，返回最终结果。

3 参数传递机制分析

在 x86-64 架构下，遵循 System V ABI 调用约定：

- 参数传递

- 若结构体大小超过寄存器容量（如 16 字节），参数通过栈传递。
- 示例中 `calc(Data a, Data b)` 的两个参数均为大结构体，因此调用者将 *a* 和 *b* 的内容依次压入栈中。
- 汇编代码中，通过 `rbp + 偏移量` 访问参数成员，例如 `movl 16(%rbp), %edx`，访问 *a* 的首地址。

- 返回值传递

- 大结构体无法通过寄存器返回，调用者需预先分配内存，并将该内存地址作为隐藏的第一个参数（通过 `%rdi` 寄存器）传递给函数。
- 函数内部将结果写入该地址，最后返回该地址。
- 示例中 `calc` 函数的汇编代码：`movq %rdi, -24(%rbp)` 保存返回地址到栈；`movq -24(%rbp), %rax` 加载返回地址；`movl %edx, (%rax)` 写入结果到返回结构体的 *v* 成员。

4 汇编代码关键片段解析

4.1 `calc` 函数参数访问

从代码中可以看到，两个结构体参数在栈上连续存放，通过固定偏移量访问成员。*a* 的起始位置是 `16(%rbp)`，*b* 的起始位置是 `80(%rbp)`。

对应代码：

```
movl 16(%rbp), %edx
movl 80(%rbp), %eax
```

4.2 返回值写入（以成员 *re* 的计算为例）

首先获取调用者传入的返回地址：`movq -24(%rbp), %rax`

然后将计算结果写入返回结构体的 *re* 成员（偏移 8 字节）：`movsd %xmm0, 8(%rax)`

从中也可以发现 *re* 成员在结构体对象首地址往后偏移 8 字节的位置。

4.3 `foo` 函数调用 `calc` 时传参的过程

首先，在栈上分配存放参数的空间：`subq $64, %rsp`

然后，将栈顶地址作为参数起始地址：`movq %rsp, %rax`

然后，拷贝结构体成员到栈：`movq %rcx, (%rax)`

传递返回地址：`movq %rdx, %rdi`

调用 `calc` 函数：`call calc`

综上，调用者需手动拷贝结构体内容到栈，并传递返回地址。