```
 9  }
10
11  double complex c_sub(double complex x, double complex y) {
12      return x - y;
13  }
```

编译时，GCC 为这些函数产生如下代码：

```
    double c_imag(double complex x)
1   c_imag:
2     movapd  %xmm1, %xmm0
3     ret

    double c_real(double complex x)
4   c_real:
5     rep; ret

    double complex c_sub(double complex x, double complex y)
6   c_sub:
7     subsd   %xmm2, %xmm0
8     subsd   %xmm3, %xmm1
9     ret
```

根据这些例子，回答下列问题：

A. 如何向函数传递复数参数？

B. 如何从函数返回复数值？

练习题答案

3.1　这个练习使你熟悉各种操作数格式。

操作数	值	注释
%rax	0x100	寄存器
0x104	0XAB	绝对地址
$0x108	0x108	立即数
(%rax)	0XFF	地址 0x100
4(%rax)	0XAB	地址 0x104
9(%rax,%rdx)	0x11	地址 0x10C
260($rcx,%rdx)	0x13	地址 0x108
0XFC(,%rcx,4)	0xFF	地址 0x100
(%rax,%rdx,4)	0x11	地址 0x10C

3.2　正如我们已经看到的，GCC 产生的汇编代码指令上有后缀，而反汇编代码没有。能够在这两种形式之间转换是一种很重要的需要学习的技能。一个重要的特性就是，x86-64 中的内存引用总是用四字长寄存器给出，例如%rax，哪怕操作数只是一个字节、一个字或是一个双字。

这里是带后缀的代码：

```
movl   %eax, (%rsp)
movw   (%rax), %dx
movb   $0xFF, %bl
movb   (%rsp,%rdx,4), %dl
movq   (%rdx), %rax
movw   %dx, (%rax)
```

3.3　由于我们会依赖 GCC 来产生大多数汇编代码，所以能够写正确的汇编代码并不是一项很关键的技能。但是，这个练习会帮助你熟悉不同的指令和操作数类型。