这段代码会打印出：

```
a = 3, b = 4
```

C 操作符 &(称为“取址”操作符)创建一个指针，在本例中，该指针指向保存局部变量 a 的位置。然后，函数 exchange 将用 3 覆盖存储在 a 中的值，但是返回原来的值 4 作为函数的值。注意如何将指针传递给 exchange，它能修改存在某个远处位置的数据。

练习题 3.5　已知信息如下。将一个原型为

```
void decode1(long *xp, long *yp, long *zp);
```

的函数编译成汇编代码，得到如下代码：

```
  void decode1(long *xp, long *yp, long *zp)
  xp in %rdi, yp in %rsi, zp in %rdx
decode1:
  movq    (%rdi), %r8
  movq    (%rsi), %rcx
  movq    (%rdx), %rax
  movq    %r8, (%rsi)
  movq    %rcx, (%rdx)
  movq    %rax, (%rdi)
  ret
```

参数 xp、yp 和 zp 分别存储在对应的寄存器%rdi、%rsi 和%rdx 中。

请写出等效于上面汇编代码的 decode1 的 C 代码。

3.4.4　压入和弹出栈数据

最后两个数据传送操作可以将数据压入程序栈中，以及从程序栈中弹出数据，如图 3-8 所示。正如我们将看到的，栈在处理过程调用中起到至关重要的作用。栈是一种数据结构，可以添加或者删除值，不过要遵循“后进先出”的原则。通过 push 操作把数据压入栈中，通过 pop 操作删除数据；它具有一个属性：弹出的值永远是最近被压入而且仍然在栈中的值。栈可以实现为一个数组，总是从数组的一端插入和删除元素。这一端被称为栈顶。在 x86-64 中，程序栈存放在内存中某个区域。如图 3-9 所示，栈向下增长，这样一来，栈顶元素的地址是所有栈中元素地址中最低的。(根据惯例，我们的栈是倒过来画的，栈“顶”在图的底部。)栈指针%rsp 保存着栈顶元素的地址。

指令	效果	描述
pushq　*S*	R[%rsp]←R[%rsp]−8; M[R[%rsp]]←*S*	将四字压入栈
popq　*D*	*D*←M[R[%rsp]]; R[%rsp]←R[%rsp]+8	将四字弹出栈

图 3-8　入栈和出栈指令

pushq 指令的功能是把数据压入到栈上，而 popq 指令是弹出数据。这些指令都只有一个操作数——压入的数据源和弹出的数据目的。

将一个四字值压入栈中，首先要将栈指针减 8，然后将值写到新的栈顶地址。因此，指令 pushq %rbp 的行为等价于下面两条指令：