```
/* Set all diagonal elements to val */
void fix_set_diag_opt(fix_matrix A, int val) {
    int *Abase = &A[0][0];
    long i = 0;
    long iend = N*(N+1);
    do {
        Abase[i] = val;
        i += (N+1);
    } while (i != iend);
}
```

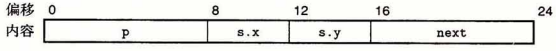
这个函数引入了一个变量 Abase，int * 类型的，指向数组 A 的起始位置。这个指针指向一个 4 字节整数序列，这个序列由按照行优先顺序存放的 A 的元素组成。我们引入一个整数变量 index，它一步一步经过 A 的对角线，它有一个属性，那就是对角线元素 i 和 $i+1$ 在序列中相隔 $N+1$ 个元素，而且一旦我们到达对角线元素 N(索引为 $N(N+1)$)，我们就超出了边界。

实际的汇编代码遵循这样的通用格式，但是现在指针的增加必须乘以因子 4。我们将寄存器%rax 标记为存放值 index4，等于 C 版本中的 index，但是使用因子 4 进行伸缩。对于 $N=16$，我们可以看到对于 index4 的停止点会是 $4 \cdot 16(16+1)=1088$。

```
1   fix_set_diag:
    void fix_set_diag(fix_matrix A, int val)
    A in %rdi, val in %rsi
2     movl    $0, %eax                  Set index4 = 0
3   .L13:                               loop:
4     movl    %esi, (%rdi,%rax)         Set Abase[index4/4] to val
5     addq    $68, %rax                 Increment index4 += 4(N+1)
6     cmpq    $1088, %rax               Compare index4: 4N(N+1)
7     jne     .L13                      If !=, goto loop
8     rep; ret                          Return
```

3.41　这个练习让你思考结构的布局，以及用来访问结构字段的代码。该结构声明是书中所示例子的一个变形。它表明嵌套的结构的分配是将内层结构嵌入到外层结构之中。

A. 该结构的布局图如下：



B. 它使用了 24 个字节。

C. 同平时一样，我们从给汇编代码加注释开始：

```
    void sp_init(struct prob *sp)
    sp in %rdi
1   sp_init:
2     movl    12(%rdi), %eax    Get sp->s.y
3     movl    %eax, 8(%rdi)     Save in sp->s.x
4     leaq    8(%rdi), %rax     Compute &(sp->s.x)
5     movq    %rax, (%rdi)      Store in sp->p
6     movq    %rdi, 16(%rdi)    Store sp in sp->next
7     ret
```

由此可以产生如下 C 代码：

```
void sp_init(struct prob *sp)
{
    sp->s.x   = sp->s.y;
    sp->p     = &(sp->s.x);
    sp->next  = sp;
}
```

3.42　这道题说明了一个非常普通的数据结构和对它的操作时如何在机器代码中实现。要解答这些问题，还是先对汇编代码加注释，确认出该结构的两个字段分别在偏移量 0(字段 v)和 8(字段 p)处。