3.59

```
store_prod:
movq %rdx, %rax  # R[%rax] = y
cqto             # R[%rdx] = yh , R[%rax] = yl
movq %rsi, %rcx  # R[%rcx] = xl
sarq $63, %rcx   # R[%rcx] = xh
imulq %rax, %rcx # R[%rcx] = xh * yl
imulq %rsi, %rdx # R[%rdx] = yh * xl
addq %rdx, %rcx  # R[%rcx] = xh * yl + yh + xl
mulq %rsi        # R[%rdx]:R[%rax] = (2^64 * yh + yl)*xl
addq %rcx, %rdx  # pl = R[%rcx] + R[%rdx]
movq %rax, (%rdi)# (%rdi) = pl
movq %rdx, 8(%rdi)# (%rdi + 8) = ph
ret
```

3.60

```
long cread_alt(long *xp)
{
    //用变量 temp 代替原函数中的常量 0，使得在 xp 为空地址时也有返回值。
    int temp = 0;
    int *p = xp ? xp : &temp;
    return *p;
}
```

3.63

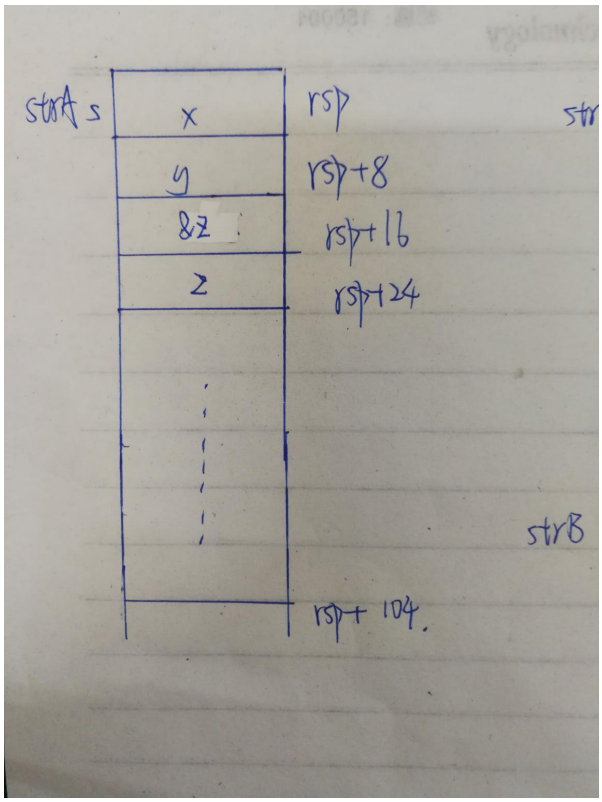
```
switch(n)
{
    case 60:
    case 62: result = 8 * x;
             break;
    case 63: result = 225*x^2 + 0x4b;
             break;
    case 64: result = x^2 + 0x4b;
             break;
    case 61:
    default: result = x + 0x4b;
}
```

3.65

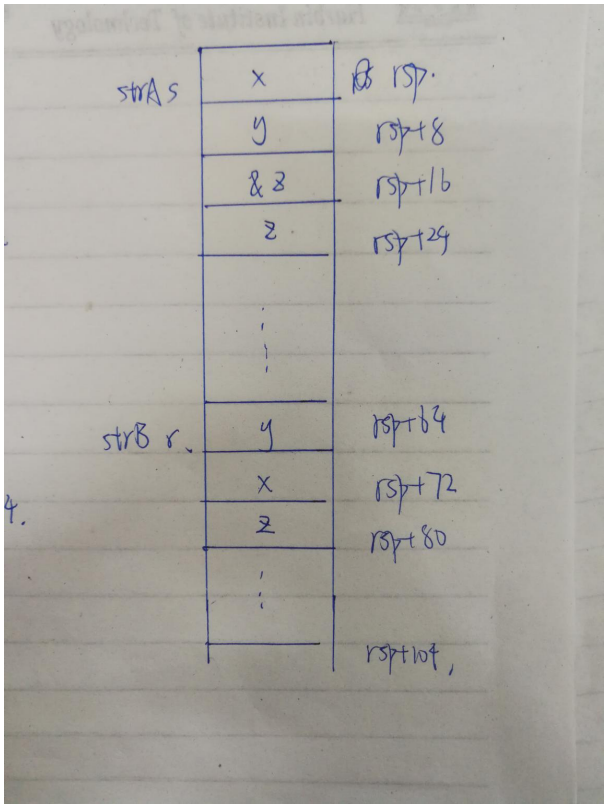
A. %rdx,由 addq $8, %rdx 知;
B. %rax,由 addq $120, %rax 知;
C. M = 120/8 = 15.

3.67

A.



B. strA s;
C. 通过将%rsp 中存储的地址变换来访问 eval 栈帧中的内容;
D. 同设置 eval 函数值设置 strA s 一样，通过%rsp 在 eval 栈帧中设置;
E. 通过%rsp 在 eval 栈帧中访问;



F. 需要调用函数分配足够大的栈帧空间并且调用函数与被调用函数同时在调用函数的栈帧中设置结构体。

3.71

```
void good_echo()
/*分块读入以实现对应任何输入长度均能工作都不会发生溢出!! */
 {
     const int Size = 0x8;
     char str[Size];
     int i;
     while(fgets(buf, Size, stdin)!=NULL)//读入过程还未完成!
     {
         for(i=0;str[i];i++)
            putchar(str[i]);
         if(i<Size-1)//读入结束判断
            break;
     }
     return;
 }
```