在 x86-64、LINUX 操作系统下，考虑如下的 C 定义:

```
typedef union{
    char c[7];
    short h;
}union_e;
typedef struct f{
    char d[3];
    union e u;
    int i;
}struct_e;
struct_e s;
```

回答如下问题:

(1) s.u.c 的首地址相对 s 的首地址的偏移量是___4____字节。

(2) sizeof(union_e)=____8____字节。

(3) s.i 的首地址相对于 s 的首地址的偏移量是____12___字节。

(4) sizeof(struct_e)=____16____字节。

(5) 若只将 i 的类型改成 short，那么 sizeof(struct_e)=___14____字节。

(6) 若只将 h 的类型改成 int，那么 sizeof(union_e) =___8____字节。

(7) 若将 i 的类型改成 short、将 h 的类型改成 int，那么 sizeof(union_e)=_____8___字节，sizeof(struct_e) =_____16____字节。

(8) 若只将 short h 的定义删除，那么(1)~(4)问的答案分别是___3___，____7____，___12_____，___16____。

解答：本题考查结构体和联合中变量的对齐，对齐的基本原则是任何 K 字节的基本对象的地址必须是 K 的倍数(P189)，解这类题目可以画一下示意图，更加直观。1~4 如下图所示：

D1	D2	D3		h	h							i	i	i	i
				C1	C2	C3	C4	C5	C6	C7					

由对齐原则，s.u.h 相对于首地址的偏移量应该是 2 的倍数，因此（1）答案是 4 而不是 3,(2)中考虑连续两个 u，第二个 u 的 h 要满足 2 的倍数，u 末尾应该有一个偏移，因此答案为 8 而不是 7。（3）（4）两问直接由图可知。

h	h							h	h						
C1	C2	C3	C4	C5	C6	C7		C1	C2	C3	C4	C5	C6	C7	

(5)如下图所示，共 14 字节，注意要求是“K 字节的基本对象”，union 不是“基本对象”，不需要满足地址是 8 的倍数。

D1	D2	D3		h	h							i	i
				C1	C2	C3	C4	C5	C6	C7			

（6）如图所示，注意问的是 union_e，不要上面几问问 struct_e 就想当然。

D1	D2	D3		h	h	h	h					i	i	i	i
				C1	C2	C3	C4	C5	C6	C7					

(7)如下图，第一空同（6），注意第二个 struct_e 中 h 的对齐要求，因而第二空答案是 16 而不是 14

D	D	D		h	h	h	h					i	i			D	D	D		h	h	h	h					i	i		

1	2	3													1	2	3											
				C1	C2	C3	C4	C5	C6	C7									C1	C2	C3	C4	C5	C6	C7				

（8）如下图，满足对齐要求（此时 union_e 等价于 char c[7]）

D1	D2	D3	C1	C2	C3	C4	C5	C6	C7			i	i	i	i

请分析下面的 C 语言程序和对应的 x86-64 汇编代码。

1.其中，有一部分缺失的代码（用标号标出），请在标号对应的横线上填写缺失的内容。注:汇编与机器码中的数字用 16 进制数填写。(1-11 每空 1 分，共 11 分)

C 语言代码如下：

```
typedef struct _parameters {
    int n;
    int product;
} parameters;
int bar(parameters *params, int x) {
    params->product *= x;
}
void foo (parameters *params) {
    if (params->n <= 1)
    ___（1）___                                          （1）_return;____
    bar(params, ___（2）___ );                           （2）_params->n__
    params->n--;
    foo(params);
}
```

x86-64 汇编代码如下（为简单起见，函数内指令地址只给出后四位，需要时可补全）：(

```
0x0000555555555189 <bar>:
5189: f3 0f 1e fa endbr64
518d: 55 push %rbp
518e: 48 89 e5 mov %rsp,%rbp
5191: 48 89 7d f8 mov _（3）_,-0x8(%rbp)             （3）_%rdi_____
5195: 89 75 f4 mov %esi,-0xc(%rbp)
5198: 48 8b 45 f8 mov -0x8(%rbp),%rax
519c: 8b 40 04 mov 0x4(%rax),%eax
519f: 0f af 45 f4 imul _（4）_(%rbp),%eax            （4）__-0xc_____
51a3: 89 c2 mov %eax,%edx
51a5: 48 8b 45 f8 mov -0x8(%rbp),%rax
51a9: 89 50 04 mov %edx,0x4(%rax)
51ac: 90 nop
51ad: 5d pop _（5）_                                 （5）__%rbp_____
51ae: c3 retq
00005555555551af <foo>:
51af: f3 0f 1e fa endbr64
```

```
51b3: 55 push %rbp
51b4: 48 89 e5 mov %rsp,%rbp
51b7: 48 83 ec 10 _（6）_ $0x10,%rsp                    （6）__sub_____
51bb: 48 89 7d f8 mov %rdi,-0x8(%rbp)
51bf: 48 8b 45 f8 mov -0x8(%rbp),%rax
51c3: 8b 00 mov (%rax),%eax
51c5: 83 f8 01 cmp $0x1,%eax
51c8: 7e 31 _（7）_ 51fb <foo+0x4c>                     （7）__jle_____
51ca: 48 8b 45 f8 mov -0x8(%rbp),%rax
51ce: 8b 10 mov (%rax),%edx
51d0: 48 8b 45 f8 mov -0x8(%rbp),%rax
51d4: 89 d6 mov %edx,%esi
51d6: 48 89 c7 mov %rax,%rdi
51d9: e8 ab ff ff ff callq 0x0000555555555189 <bar>
51de: 48 8b 45 f8 mov -0x8(%rbp),%rax
51e2: 8b 00 mov (%rax),%eax
51e4: 8d 50 ff lea -0x1(_（8）_),%edx                   （8）___%rax_____
51e7: 48 8b 45 f8 mov -0x8(%rbp),%rax
51eb: 89 10 mov _（9）_,(%rax)                          （9）___%edx_____
51ed: 48 8b 45 f8 mov _（10）_ ,%rax                    （10）_-0x8(%rbp)
51f1: 48 89 c7 mov %rax,%rdi
51f4: e8 b6 ff ff ff callq _（11）_      （11）0x00005555555551af<foo>
51f9: eb 01 jmp 51fc <foo+0x4d>
51fb: 90 nop
51fc: c9 leaveq
51fd: c3 retq
```

2.在程序执行到 0x000055555555518e 时（该指令还未执行），此时的栈帧如下，请填写空格中对应的值。（每空 2 分，共 6 分）

地址 值

```
0x7fffffffe308 0xffffe340
0x7fffffffe304 0x00000000
0x7fffffffe300 0x00000000
0x7fffffffe2fc 0x00005555
0x7fffffffe2f8                                          （12）0x555551f9
0x7fffffffe2f4 0x00007fff
0x7fffffffe2f0 0xffffe310
0x7fffffffe2ec 0x00007fff
0x7fffffffe2e8 0xffffe340
0x7fffffffe2e4 0x00000004
0x7fffffffe2e0 0xffffe350
0x7fffffffe2dc 0x00005555
0x7fffffffe2d8                                          （13）0x555551de
0x7fffffffe2d4 0x00007fff
0x7fffffffe2d0                                          （14）0xffffe2f0
```

3.当 params={n,1}时，foo(&params)函数的功能是什么？（3 分）计算阶乘
解析：建议从汇编代码开始看，汇编代码与 C 代码互为印证。

（1）考查对 51c8 及 51fb 的理解，由 51c8 和 51fb 可以知道这是跳转指令，跳转到 51fb，阅读汇编代码可知这里就结束了。结合源代码这里是 return 。

（2）考查对 51ca 和 51ce 的理解，由 51ca 可知此时 rax 保存的是 param 的地址，51ce
直接从地址取值，所以取出的是第一个数 n，即 params->n

（3）中我们看到 5191 与 5195 两个语句用以传递参数,后一个语句使用%esi，由于前一个语句传递的参数是指针，应该是 8 字节，故答案为%rdi。

（4）由 519c 可知 eax 保存的是 param->product，所以这里是执行乘法操作，另一个操作数是参数 x，5195 处已经把 x 保存在-0xc(%rbp)中，所以就从这里读取数据，即答案为-0xc。

（5）结合 518d 可知答案。

（6）进入函数后首先分配栈帧，通过减%rsp 实现，故这里填 sub。

（7）结合 C 程序中的 if(params->n<=1)可知,这里应该是如果 n<=1 则执行跳转，应该是 jle。同时我们发现跳转到 51fb 之后什么都不做直接返回，可知（1）答案为 return。

（8）51e2 已经将 params->n 读入到%eax 中，结合源程序中 params->n--，这里进行修改，应该填%rax。

（9）还是考查对 params->n--的理解，上一空是将计算结果保存到%edx 里，这一步要将结果存回（%rax），因此这一空填%edx。

（10）考查函数的传参方法，与 51e7 相同。

（11）Foo 函数的地址，注意不要抄错了。

（12）结合前面的 sub $0x10,%rsp,知有 0x10 的大小都是 foo 的栈帧。这里应该恰好是 foo 的返回地址，即调用 foo 之后下一条指令的地址。

（13）由于 518e 在 bar 函数哪里，在 foo 函数中进入 bar 函数，所以这里应该是 bar 函数返回地址，返回的是调用 bar 后的下一条指令地址。

（14）这一空是 Push 之后栈顶的内容，推测出此时的 rbp 是在上一次函数调用，进入 bar 时设置的。上一次保存函数调用的返回地址在 0x7fffffffe2f8，所以上一次 rbp 的值是 0x7fffffffe2f0。

最后一问由于初始时 product 为 1，每次乘 n，递归中 n 每次减一，n<=1 停止，知计算的是 n 阶乘。

阅读下面的汇编代码：

```
<f>:
  4004c4:       push   %rbp
  4004c5:       mov    %rsp,%rbp
  4004c8:       sub    $0x10,%rsp
  4004cc:       mov    %edi,-0x4(%rbp)

  4004cf:       cmpl   $0x1,-0x4(%rbp)
  4004d3:       ja     4004dc <f+0x18>
  4004d5:       mov    $0x1,%eax
```

```
4004da:       jmp    40052d <f+0x69>
4004dc:       mov    -0x4(%rbp),%eax
4004df:       and    $0x1,%eax

4004e2:       test   %eax,%eax
4004e4:       jne    4004f5 <f+0x31>
4004e6:       mov    0x200440(%rip),%eax        # 60092c <x.1604>
4004ec:       add    $0x1,%eax
4004ef:       mov    %eax,0x200437(%rip)        # 60092c <x.1604>
4004f5:       mov    -0x4(%rbp),%eax
4004f8:       and    $0x1,%eax

4004fb:       test   %al,%al
4004fd:       je     40050e <f+0x4a>
4004ff:       mov    0x20042b(%rip),%eax        # 600930 <y.1605>
400505:       add    $0x1,%eax
400508:       mov    %eax,0x200422(%rip)        # 600930 <y.1605>
40050e:       mov    -0x4(%rbp),%eax
400511:       sub    $0x1,%eax
400514:       mov    %eax,%edi
400516:       callq  4004c4 <f>
40051b:       mov    0x20040f(%rip),%edx        # 600930 <y.1605>
400521:       lea    (%rax,%rdx,1),%edx
400524:       mov    0x200402(%rip),%eax        # 60092c <x.1604>
40052a:       lea    (%rdx,%rax,1),%eax
40052d:       leaveq
40052e:       retq
```

1）程序

```
main()
{
    unsigned int n;
    for (n=1; n< 4; n++) {
        printf("f(%d) = %x\n", n, f(n));
    }

}
```

的运行结果为：f(1)=1，f(2)=4e，f(3)=9f，请填写 f 函数所需要的内容（每空 1 分）：

```
#define N   (1)             3
#define M   (2)             5
```

```
struct P1 {char c[N]; char *d[N]; char e[N]; } P1;
struct P2 {int i[M]; char j[M]; short k[M]; } P2;

unsigned int f(unsigned int n)
{
     (3)static                  unsigned int x = sizeof(P1);
     (4)static                  unsigned int y = sizeof(P2);

    if (  (5) n<=1                    )
        return 1;

    if (  (6) (n&1)==0                    )
        x++;

    if (  (7) (n&1)!=0                    )
        y++;

    return  (8)f(n-1)+x+y                    ;
}
```

2、程序

```
main()
{
    printf("%x, %x\n", f(2), f(2));
}
```

的运行结果为：（2 分）4f,4e

这种问题考察对汇编代码的阅读与理解。建议结合汇编语言与 C 语言互为印证，先从（5）（6）（7）（8）开始解答。阅读汇编代码的时候建议根据比较和跳转语句对其进行初步的划分（当然这种划分不见得精确，交界部分语句属于上一部分还是下一部分要理解了汇编代码才能准确确定）。

考虑 cmpl　$0x1,-0x4(%rbp)与 ja　4004dc <f+0x18>两句，结合再前面一句知道-0x4(%rbp)存放的是变量 n，ja 语句之后的 mov　$0x1,%eax 表明了如果不发生跳转，就直接将 1 作为答案返回，那么（5）答案为 n<=1。

结合 and　$0x1,%eax 与 test　%eax,%eax 与 jne　4004f5 <f+0x31>三句话可知，如果 x&1 不为 0 将执行跳转，再考虑 mov　0x200440(%rip),%eax 和 add　$0x1,%eax 和 mov　%eax,0x200437(%rip)可知，不发生跳转结果就会执行 x++，那么（6）应该为（n&1）==0，注意运算顺序，不清楚的时候可以加上括号以防万一，同理（7）应该为（n&1）!=0。

考虑 callq 4004c4 <f>与之后几句话，%eax 中存的是 f(n-1),这几句运算是计算 f(n-1)+x+y 的值并且返回，那么(8)答案即为 f(n-1)+x+y.

回到（3）（4），注意到 f(3)-f(2)不等于 f(2)-f(1)，如果对于每次调用 f，均产生新的 x 与 y，那么无论执行 x++或者 y++，两者都应该相等，可见 x 与 y 为所有的 f 公用，那么应该是静态变量，答案为 static。

再看（1）与（2），考虑 f(2)与 f(1)的差值，得到 x+y=76，穷举 m 和 n 所有可能值，得 n=3，m=5。（这一部分我没有想到很合适的解法，因为对齐的缘故并不容易将 x 与 y 直接写成 n 和 m 的函数，只好穷举）。

最后看第二大问，难点在于知道 printf 执行的时候先计算后一个 f(2)，另外注意作答规范，不要少了中间的“,”，也不要在答案之前加上“0x”。