1. 在 x86 机器上有如下代码 “ int a=15;”且已知 a 的地址为 0x40030，请说明存储变量 a 需要的字节数 n，以及从 0x40030 开始的 n 个字节上存储什么内容？如果在 IBM 的 power8 处理器这样的大端机器上，各个字节又是什么内容？

答：1) n=4; 2) 依次为 0x0F 0x00 0x00 0x00; 3) 依次为 0x00 0x00 0x00 0x0F

2. 请判定 C 语言表达式 -2147483647 – 1U < 2147483647 取值。

答：取值为 0（false）

3. 有如下代码：char x = -8; unsigned int y = x；请问 y 的二进制位模式是什么？数值为多少？

答：二进制位模式为 0xf8 ff ff ff (或 0xff ff ff f8)；数值为 2^32-8

4. 有以下代码 unsigned int x = 15; int y = -4；x>>=3;y>>=3; 请问最后 x 和 y 的数值为多少？

答：x=1, y=-1

5. 有以下代码 unsigned char x=128; x=x+x; 请问 x 最后的二进制位模式是什么？

二进制位模式为 0x00

6. 写出下面函数 Func1 汇编代码对应的 C 程序，其中参数 1 为 x，参数 2 为 y：

```
Func1:
        cmpq    %rsi, %rdi
        jge     .L2
        leaq    3(%rsi), %rdi
        jmp     .L3
.L2:
        leaq    (%rdi,%rdi,4), %rsi
        addq    %rsi, %rsi
.L3:
        leaq    (%rdi,%rsi), %rax
        ret
```

答：

```
// %rdi = x, %rsi = y
long Func(long x, long y) {
   if (x < y) {
        x = 3 + y;          // return 2y+3
   } else {
        y = 5*x;
        y += y; // return 11y
   }
   return x + y;
}
```

7. 观察以下 C 代码和对应的 x86-64 汇编代码，其中 H 和 J 是由#define 声明的常量

```
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
   array2[x][y] = array1[y][x];
}
---------------------------------
# On entry:
# %edi = x
# %esi = y
#
copy_array:
   movslq      %esi,%rsi
   movslq      %edi,%rdi
   movq        %rdi, %rax
   salq        $4, %rax
   subq        %rdi, %rax
   addq        %rsi, %rax
   leaq        (%rsi,%rsi,4), %rsi
   leaq        (%rdi,%rsi,2), %rsi
   movl        array1(,%rsi,4), %edx
   movl        %edx, array2(,%rax,4)
   ret
```

H 和 J 的值是多少？

H = 15

J = 10

8. 观察以下 x86-64 汇编代码

```
loop:
# on entry: a in %rdi, n in %esi
   movl     $0, %r8d
   movl     $0, %ecx
   testl    %esi, %esi
   jle      .L3
.L6:
   movl     (%rdi,%rcx,4), %edx
   leal     3(%rdx), %eax
   testl    %edx, %edx
   cmovns   %edx, %eax
   sarl     $2, %eax
   addl     %eax, %r8d
   addq     $1, %rcx
   cmpl     %ecx, %esi
   jg       .L6
.L3:
   movl     %r8d, %eax
   ret
```

请根据汇编代码，补全下面 C 代码中的空白部分，请注意：

- 变量名只能使用 n，a，i 和 sum 其中之一，不能直接使用寄存器名
- 使用数组符号访问或更新 a 中的元素

```
int loop(int a[], int n)
{
   int i, sum;
   sum = __0__;
   for (i = __0__ ; ___n___ ; ____i++_____) {
      sum += _____a[i]/4  or  (a[i] < 0 ? a[i] + 3 : a[i]) >> 2___;
   }
   return  _____sum_____;
}
```

9. 下面给出了四个 C 函数和四个 x86-64 代码块。在每个 x86-64 代码块的旁边，写它实现的 C 函数的名称。

```
int alpha(struct node *ptr) {
   return ptr->d.x;
}
char *beta(struct node *ptr) {
   ptr = ptr->next;
   return ptr->d.str;
}
char gamma(struct node *ptr) {
   return ptr->d.str[7];
}
long *delta(struct node *ptr) {
   struct data *dp = (struct data *) ptr;
   return &dp->x;
}
char *epsilon(struct node *ptr) {
   return &ptr->d.str[2];
}
```

gamma	movsbl 15(%rdi),%eax ret
alpha	movq (%rdi), %rax ret

beta	movq 24(%rdi), %rax
	addq $8, %rax
	ret

delta	movq %rdi, %rax ret
epsilon	leaq 10(%rdi), %rax ret

其中，data 和 node 结构的声明如下：

```
struct data {
   long x;
   char str[16];
};
```

```
struct node {
   struct data d;
   struct node *next;
};
```

10. 用适当的表达式填充 lmao 的 C 代码中缺失的部分。（注：0x400498 是 C 标准库函数 malloc 的地址。）

```
typedef struct node
{
        void *data;
        struct node *next;
} node_t;

node_t *lmao(node_t *n, int f(node_t *))
{
        node_t *a, *b;

        if( !n )
        {
                return NULL;
        }

        a = lmao(n->next, f);

        if( f(n) > 0 )
        {
                b = malloc(16);
                b->data = n->data;
                b->next = a;
                return b;
        }

        return a;
}
```

```
0x4005d0:    mov    %rbx,-0x18(%rsp)
0x4005d5:    mov    %rbp,-0x10(%rsp)
0x4005da:    xor    %eax,%eax
0x4005dc:    mov    %r12,-0x8(%rsp)
0x4005e1:    sub    $0x18,%rsp
0x4005e5:    test   %rdi,%rdi
0x4005e8:    mov    %rdi,%rbx
0x4005eb:    mov    %rsi,%rbp
0x4005ee:    je     0x40061e <lmao+78>
0x4005f0:    mov    0x8(%rdi),%rdi
0x4005f4:    callq  0x4005d0 <lmao>
0x4005f9:    mov    %rbx,%rdi
0x4005fc:    mov    %rax,%r12
0x4005ff:    callq  *%rbp
0x400601:    mov    %eax,%edx
0x400603:    mov    %r12,%rax
0x400606:    test   %edx,%edx
0x400608:    jle    0x40061e <lmao+78>
0x40060a:    mov    $0x10,%edi
0x40060f:    callq  0x400498 <malloc>
0x400614:    mov    (%rbx),%rdx
0x400617:    mov    %r12,0x8(%rax)
0x40061b:    mov    %rdx,(%rax)
0x40061e:    mov    (%rsp),%rbx
0x400622:    mov    0x8(%rsp),%rbp
0x400627:    mov    0x10(%rsp),%r12
0x40062c:    add    $0x18,%rsp
0x400630:    retq
```