

1. How many bytes would the following array declaration allocate on a 64-bit machine?

```
char *arr[10][6];
```

2. What will the following print out?

```
typedef struct {
    char shookie;
    int tata;
    char cookie;
    double chimmy;
} bt;

void main(int argc, char** argv){
    bt band[7];
    printf( "%d\n", (int)sizeof(band));
}
```

3. What is the best* ordering of the following data types if you want to have a struct that uses all of them? What is this optimal size? Assume a 64-bit architecture.
** the ordering that will result in the optimal usage of space – there's more than 1 answer!*

```
char tully;
long stark;
float* lannister;
double targaryen;
int greyjoy;
float arryn;

struct Westeros{
    /* order the above variables here */

};
```

4. Consider the following disassembled function:

```

000000000040102b <phase_2>:
  40102b:  55                push    %rbp
  40102c:  53                push    %rbx
  40102d:  48 83 ec 28       sub     $0x28,%rsp
  401031:  48 89 e6          mov     %rsp,%rsi
  401034:  e8 e3 03 00 00    callq   40141c
<read_six_numbers>
  401039:  83 3c 24 01       cmpl    $0x1, (%rsp)
  ...

```

Right after the callq instruction has been executed, what address will be at the top of the stack?

5. Consider the following C code:

```

typedef struct {
    char first;
    int second;
    short third;
    int* fourth;
} stuff;

stuff array[5];

int func0(int index, int pos, long dist) {
    char* ptr = (char*) &(array[index].____);
    ptr += pos;
    *ptr = ____ + dist;

    return *ptr;
}

int func1() {
    int x = func0(1, ____, ____);
    return x;
}

```

Clearly some code is missing - your job is to fill in the blanks! Note that the size of the blanks is not significant. The two functions will be compiled using the following assembly code:

0000000000400492 <func0>:

```
400492: 8d 04 17          lea    (%rdi,%rdx,1),%eax
400495: 48 63 ff          movslq %edi,%rdi
400498: 48 63 f6          movslq %esi,%rsi
40049b: 48 8d 14 7f       lea    (%rdi,%rdi,2),%rdx
40049f: 88 84 d6 60 10 60 00 mov    %al,0x601060(%rsi,%rdx,8)
4004a6: 0f be c0          movsbl %al,%eax
4004a9: c3               retq
```

00000000004004aa <func1>:

```
4004aa: c6 05 cb 0b 20 00 0d movb    $0xd,0x200bcb(%rip)
                                # 60107c <array+0x1c>
4004b1: b8 0d 00 00 00    mov     $0xd,%eax
4004b6: c3               retq
```