Name:	UID:

1. What's that Size?

What will be printed after running from the following code-snipet?

```
typedef struct {
    char Rumi;
    int Mira;
    char Zoey;
    double Celine;
} Huntrx;

typedef union {
    char Jinu;
    int Mystery;
    char Abby;
    double Romance;
} SajaBoys;

int main(int argc, char** argv) {
    Huntrx band1[7];
    SajaBoys band2[7];
    printf("%d\n", (int)sizeof(band1));
    printf("%d\n", (int)sizeof(band2));
    return 0;
}
```

2. Struct ordering and optimal size

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 best ordering means the one that results in optimal space usage — there's more than one valid answer!

```
1  char tully;
2  long stark;
3  int greyjoy;
4  float* lannister;
5  float arryn;
6  double targaryen;
7
8  struct Westeros {
9      /* order the above variables here */
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28 };
```

3. Tricky Switch

Reverse engineer the assembly code on the previous page to figure out what each case of the switch-case statement is doing. Don't forget about "break" statements!

Source Code (C) (fill in blanks!)

```
int func(int x, int y, int r)
    {
    switch (x) {


      case 0: ___________


      case 1: ___________


      case 2: ___________


      case 3: ___________


      case 4: ___________


      case 5: ___________
    }


    return _______________;
}
```

Compiled Assembly

```
func(int, int, int):
        cmpl    $5, %edi
        ja      .L9
        movl    %edi, %edi
        jmp     *.L4(,%rdi,8)

.L4:    #hint, this is the jump table!
        .quad   .L8
        .quad   .L7
        .quad   .L6
        .quad   .L5
        .quad   .L5
        .quad   .L3
.L7:
        addl    $6, %edx
.L6:
        leal    (%rdx,%rdx,4), %eax
        ret
.L5:
        leal    (%rdx,%rsi,2), %eax
        ret
.L3:
        movl    %edx, %eax
        xorl    %esi, %eax
        ret
.L8:
        movl    $-4, %eax
        ret
.L9:
        movl    %edx, %eax
        ret
```

4. Fill in the missing 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:

```
0000000000401106 <func0>:
  401106:    48 63 c7                movslq %edi,%rax
  401109:    48 8d 04 40             leaq   (%rax,%rax,2),%rax
  40110d:    48 63 f6                movslq %esi,%rsi
  401110:    01 d7                   addl   %edx, %edi
  401112:    40 88 bc c6 60 40 40 00 movb   %dil, 0x404060(%rsi,%rax,8)
                                              # 0x404060 <array>
  40111a:    40 0f be c7             movsbl %dil, %eax
  40111e:    c3                      retq

000000000040111f <func1>:
  40111f:     ba 0c 00 00 00         movl   $0xc,%edx
  401124:     be 04 00 00 00         movl   $0x4,%esi
  401129:     bf 01 00 00 00         movl   $0x1,%edi
  40112e:     e8 d3 ff ff ff         callq  401106 <func0>
  401133:     c3                     retq
```

What operation does this function do?