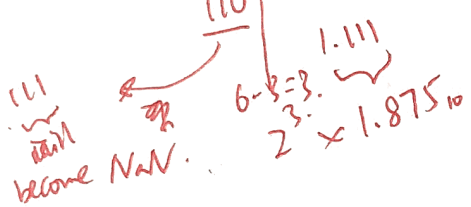
12

Question 1. The bigger the better. (18, 3 pts each)

1. Consider an n-bit signed number, what's the largest one? $2^{n-1}-1$
2. In C, what's the largest `int` plus one? Tmin
3. Which can represent the largest number in C, the largest `float` or the largest `signed long` or largest `unisgned int`? ~~largest signed long~~ largest float
4. Which integer type in C is large enough to store a pointer without loss of precision? ~~long~~
5. What is the largest number that can be represented by a 7 bit floating point number (say with the same rules as IEEE 754 floating point), with a 1 bit sign, 3 bit exponent, and 3 bit significand (bias=3)? 0 111 111 Value? $1.875 \times 2^3 = 15$
6. In C, what's the smallest `unsigned int` minus one? UMax



Question 2. Matchmaker (8 Pts, 1 pts each)

8

Pretend to be a compiler.
You are free to assign registers to variables however you choose. Assume x and y are of type int. Remember, the compiler(me) may have done some optimizations.

h	x=(x < 0) ? -1 : 0	(a) `addl %edi %edi`
c	x=x*3+5	(b) `xorl %edi %edi`
f	x=x*32	(c) `leaq 5(%edi,%edi,2)` 3edi+5
b	x=0	(d) `imul %edi %edx`
e	x=1	(e) `movl $1 %eax`
g	x=x*5+3	(f) `shl $ 5 %edi` <<5
d	x=x*y	(g) `leaq 3(%edi,%edi,4)` 4edi+edi = 5edi+3.
a	y=x+y	(h) `shr $ 31 %edi`

Question 3. Unholy Union (9 pts)

i2

i2

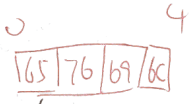
```
#include <stdio.h>
#include <string.h>

void main(char** argv, int argc) {
  union U {
    char s[16];
    int i;
    char c;
  } u;

  strcpy(u.s,"evil_prof"); //Copy string to destination from source

  printf("%x\n", u.c);
  printf("%x\n", u.i);
}
```

evil_prof — endianess doesn't matter

10

1. What does this program print? (6 pts)

0x65
0x6576696c 0x6c697665



evil

int, reads in this address

2. To which addresses may this union be aligned? (3pts)

Should be aligned to the biggest type of 4 bytes, which is int,
So total offset should be 16.

Question 4. Deconstructed (20 pts, 5 Each)

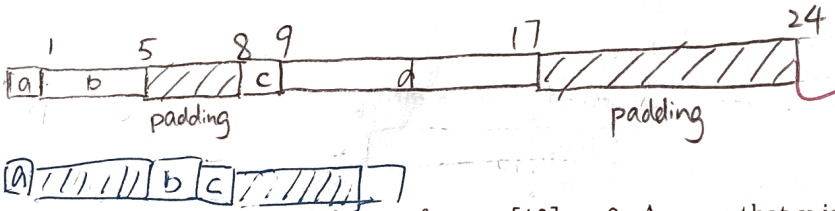
```
#include <stdio.h>

typedef struct {
 char a;
 int b;
 char c;
 double d;
} X;

void main(char** argv, int argc) {
  X x[10];
  printf("%d\n",(int)sizeof(X));
  printf("%d\n",(int)sizeof(x));
}
```

1. What does this program print?

   24

   240

2. Draw the memory layout of X, where your diagram indicates which byte offset each variable is located at, as well as any space allocated just for padding:



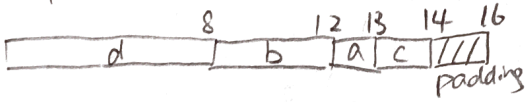
10 * 24 + 8

3. Write an assembly snippet that performs `x[10].c=0`. Assume that x is in register $rdi.

```
leaq 8(,%rdi,240), %rdi
movq $0,(%rdi)
```

4. Describe how you would reduce the memory consumption of x. How small can you make x?

By putting the largest data type in front, as shown.



X will have a size of 16.

So x will have a size of 160.

Question 5. I can puzzle, (15 Pts, 2 pts each)

Answer these true false puzzles. Assume the following setup:

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

T `-x == ~x+1`

F `x > 0 && y > 0 ⟹ x + y > 0`

F `5*ux > ux`

F `x < 100 ⟹ 10*ux > ux`

F `x >> 2 == x / 4`

Question 6. ... and so can you! (Up to 4 pts Extra Credit)

1. Write a C Puzzle of the form above, give the solution, and explain why you think its cool.

(ux * uy) == (x * y) True. +3

Since both unsigned and signed are involved, signed int are casted to unsigned implicitly.

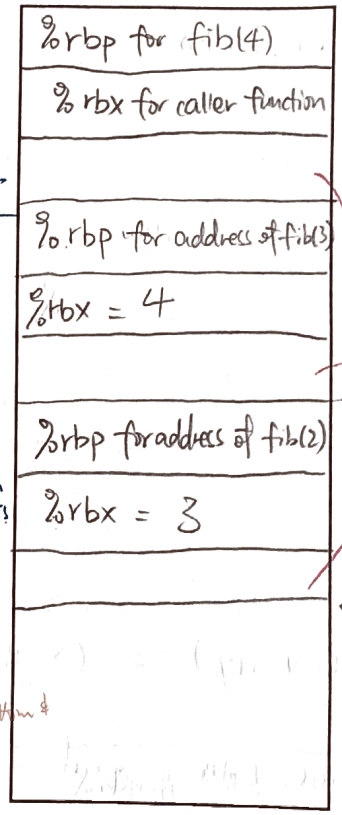
So right side becomes ux * uy, which is the same as left side

Question 7. Your fibs are stacking up (16 Pts)

Recall the fibbonacci code that we discussed in class, and its associated disassembly: (the instruction addresses are omitted for simplicity, just the offsets remain)

```
int fib(int a) {
  if(a < 2) {
    return 1;
  }
  return fib(a-1) + fib(a-2);
}
```

```
fib: 0x40055d <+0>:  push   %rbp
     0x40055e <+1>:  push   %rbx
     0x40055f <+2>:  sub    $0x8,%rsp
     0x400563 <+6>:  mov    %edi, %ebx
     0x400565 <+8>:  cmp    $0x1, %edi
     0x400568 <+11>: jle    0x400580 <fib+35>
     0x40056a <+13>: lea    -0x1(%rdi), %edi
-->  0x40056d <+16>: callq  0x40055d <fib>
     0x400572 <+21>: mov    %eax, %ebp
     0x400574 <+23>: lea    -0x2(%rbx), %edi
     0x400577 <+26>: callq  0x40055d <fib>
     0x40057c <+31>: add    %ebp, %eax
     0x40057e <+33>: jmp    0x400585 <fib+40>
     0x400580 <+35>: mov    $0x1, %eax
     0x400585 <+40>: add    $0x8, %rsp
     0x400589 <+44>: pop    %rbx
     0x40058a <+45>: pop    %rbp
     0x40058b <+46>: retq
```



1. This function calls itself recursively. Imagine in gdb we put a breakpoint on line 0x40056d, then call fib(4). Furthermore we hit continue two more times in gdb, so that the stack frames of fib(4), fib(3), and fib(2) are all on the stack. Draw the contents of the stack in the box above, and be sure to indicate the stack pointer. Draw everything you know about the stack! If you know what the value is, write the value, otherwise indicate what it is. (10 pts)

2. On which line(s) (specify as offset from fib please!) is/are callee saved registers being saved? (1pt) <+0> and <+1>

3. On which line(s) is/are callee saved registers being restored? (1pt) <+44> and <+45>

4. On which line(s) is/are the input argument to fib being set? (1pt) <+13> and <+23>

5. On which line(s) is/are the return value from fib being set (for the final time)? (1pt) <+31>

6. On which line(s) is/are the stack being allocated? (1pt) <+2>

7. On which line(s) is/are the stack being de-allocated? (1pt) <+40>

Question 8. Oh Fuuuudge (10 pts)

You just finished your CS32 homework when all of a sudden you "`rm -f my_homework.c`". Thankfully, you didn't delete your binary file – phew. You forgot all the expressions in your source code, but you kind of remembered the overall structure. It's time to analyze the binary to fill out the remaining expressions.

```
<+0>:  mov     $0x1, %r9d
<+6>:  jmp     <func+54>
<+8>:  movslq  %r9d, %rax
<+11>: mov     (%rdi, %rax, 4), %r8d
<+15>: lea     -0x1(%r9), %eax
<+19>: jmp     <func+28>
<+21>: mov     %edx, 0x4(%rdi, %rcx, 4)
<+25>: sub     $0x1, %eax
<+28>: test    %eax, %eax
<+30>: js      <func+43>
<+32>: movslq  %eax, %rcx
<+35>: mov     (%rdi, %rcx, 4), %edx
<+38>: cmp     %r8d, %edx
<+41>: jg      <func+21>
<+43>: cltq
<+45>: mov     %r8d, 0x4(%rdi, %rax, 4)
<+50>: add     $0x1, %r9d
<+54>: cmp     %esi, %r9d
<+57>: jl      <func+8>
<+59>: repz retq
```

1. Fill in the code (2 Pts each .. Extra Credit Possible)

```
void func(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i-1;

        while (j>=0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
}
```

2. What well-known algorithm is this? (2 Pts Extra Credit)

sorting algorithm

Shell sort ×

+1

insertion sort