Yeah…. It is from an exam. You can fill it in via computer or hand

UNCA CSCI 255
Final Exam Fall 2016

Problem 2 (5 points) C expressions
In the left column, there are some tricky, and some not-so tricky, C expressions. (Except for the first two, these are also Java expressions.) Write their values in the right column. Express your answers in base 10. Assume two's complement representation.

You need to do these perfectly for the next exam

`! (15 > 2)`	
`6 || (143 * 66 / 33)`	
`25 << 3`	
`25 >> 3`	
`~25`	
`25 + 15`	
`25 | 15`	
`25 ^ 15`	
`3 * 2 / 4`	
`3 + 2 / 4`	

Problem 5 (3 points) Binary arithmetic
Perform the following operations and express the results as they should be for CSCI ~~255~~ 235.

64k * 32
128M / 16
$\log_2(32k)$

Problem 6 (4 points) Adding signed numbers

Add the following pairs of five-bit *two's complement* numbers ~~**and indicate which additions result in an overflow by writing one of "overflow" or "no overflow" in each box**~~. ~~You must write either "overflow" or "no overflow" in each box in addition to the result of the addition.~~

Let's change it a little.
Remember how the x86-64 has those four bits: CF, OF, SF, ZF
How would those four bits be set by each addition?

01011 + <u>01111</u> CF__, OF__, SF__, ZF__	10111 + <u>11101</u> CF__, OF__, SF__, ZF__
10101 + <u>10101</u> CF__, OF__, SF__, ZF__	11100 + <u>01000</u> CF__, OF__, SF__, ZF__

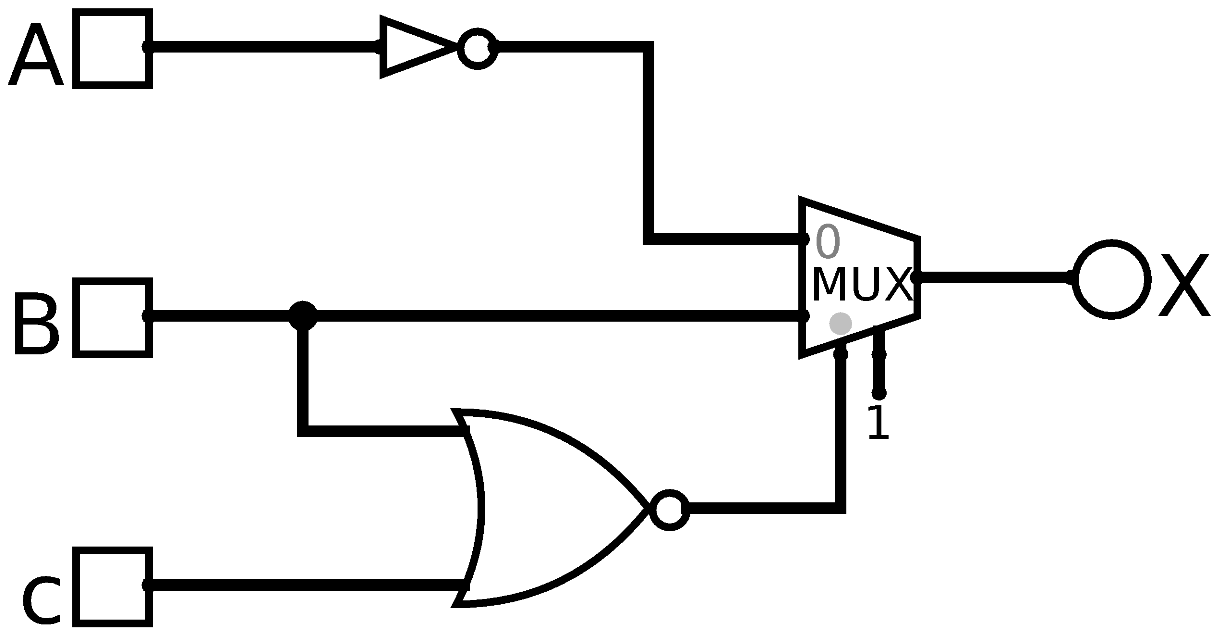
Problem 8 (3 points) Fixed point encoding

In the left column are decimal numbers. Express these numbers as five-bit twos-complement fixed point numbers with two fractional bits (and consequently three integer bits) in the right column. *You may not be able to get an exact representation for all of them, but get as close as you can.*

2.71828	
-2.75	

Problem 11 (7 points) Circuit to Boolean table and expression

Shown below is a digital circuit with inputs **A**, **B** and **C** and a single output **X**. The box labeled **MUX** is a multiplexer with two data inputs and one select input.



This circuit was generated by the logisim program. Ignore the input '1' on bottom right of the MUX. The bottom left input of the MUX is a selector bit. The two inputs on the left side are the data inputs.

First, complete the truth table for the circuit.

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Second, write a boolean equation for the circuit.

Problem 12 (7 points) Boolean expression to truth table and circuit

First, fill in the truth table on the right below so that it corresponds to the following Boolean equation

$$X = A\ B + \overline{B + C}$$

If you prefer that your negations be primes, you can think of the equation as

$$X = A\ B + (B + C)'$$

Or, if you really like Java and C expressions, you can go with

```
X = A && B || !(B || C)
```

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

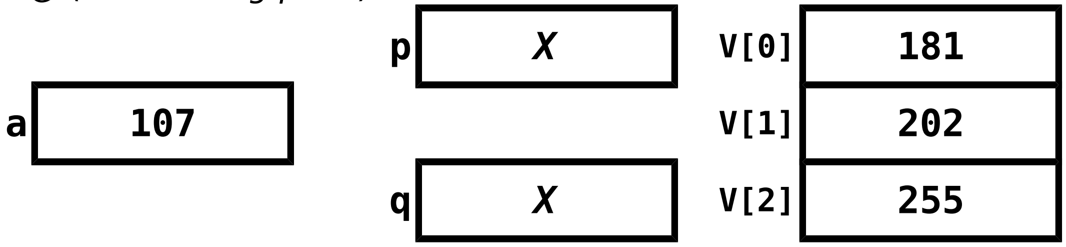
Second, draw a logic circuit to implement the boolean equation and truth table.

Problem 16 (8 points)
In this question, you are to fill in boxes representing the following C integer or pointer variables to show their values after each of seven sections of C code are executed. **You should consider all the sections as being independently executed after the following declaration and initialization statements**:

```
int    a = 107 ;
int    V[3] = {181, 202, 255} ;
int    *p = NULL ;
int    *q = NULL ;
```

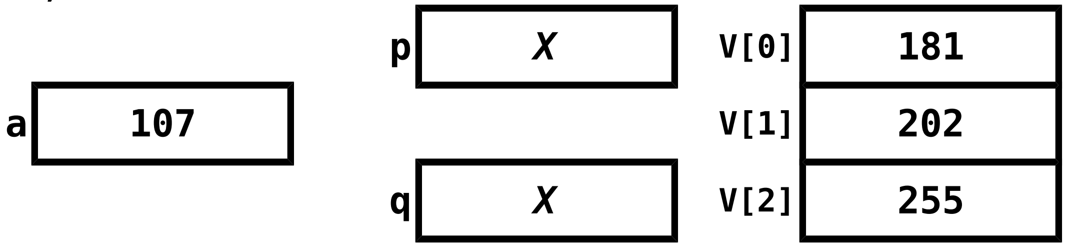
As you might guess, **`null`** in Java is similar to **`NULL`** in C. Draw the value **`NULL`** with a little ***X***. Don't ever just leave the pointer variable boxes empty.

Code section @ (the starting point)



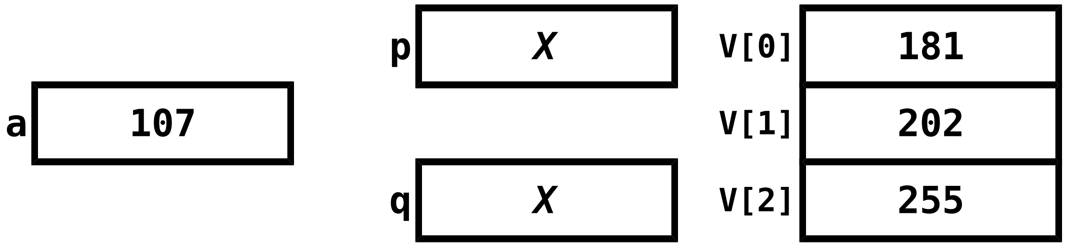
Code section A

```
p = &V[1] ;
q = &V[2] ;
*p = 200 ;
*q = 300 ;
```



Code section B

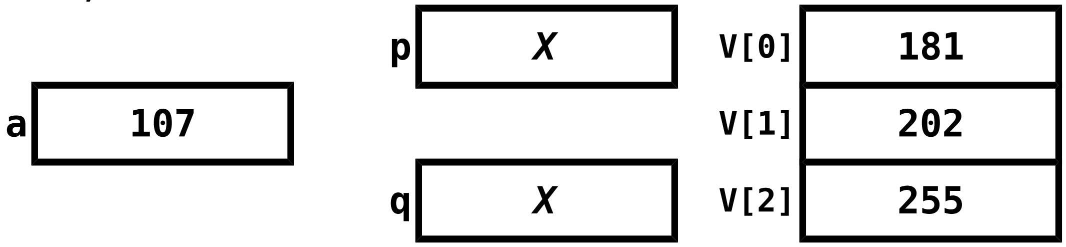
```
p = V ;
q = p + 1 ;
*p = *q ;
```



Code section C

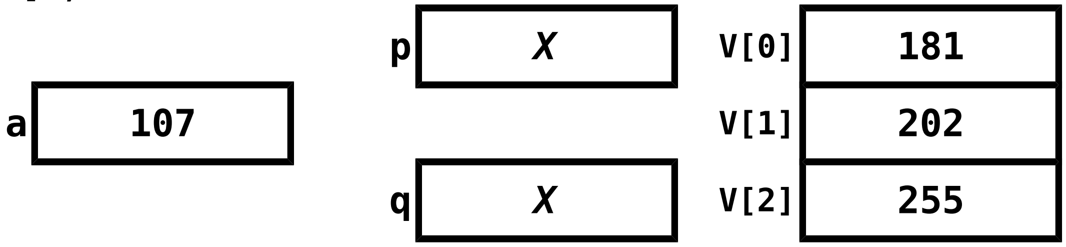
```
V[0] = a++ ;
```

```
   V[1] = a++ ;
   V[2] = ++a ;
```



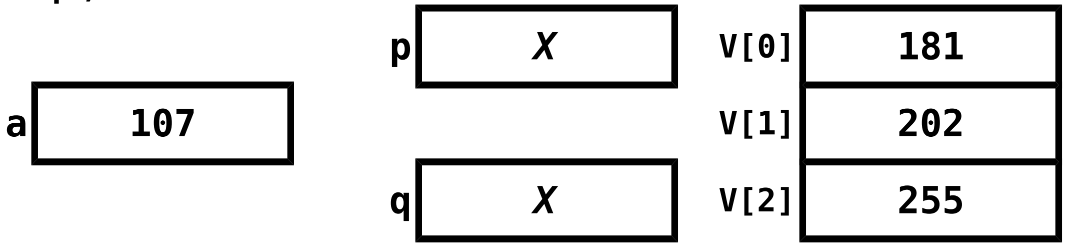
Code section D

```
   p = &a ;
   q = &V[0] ;
   *p = q[1] ;
```



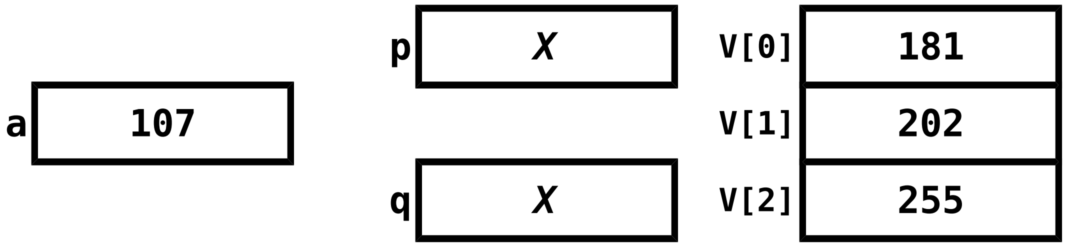
Code section E

```
   p = &V[2] ;
   q = &V[0] ;
   *p = *p – *q ;
   *q = p – q ;
```



Code section F

```
   p = &V[0] ;
   *p++ = 320 ;        // same as *(p++) = 320 ;
   a = (*p)++ ;
```



The last two problems are based on the following C function.

```
int positiveSum(int V[], int size) {
  int sum = 0 ;
  for (int i=0; i<size; ++i) {
    if (V[i] >= 0) {
      sum = sum + V[i] ;
    } else {
      V[i] = 0 ;
    }
  }
  return sum ;
}
```

You will write your answers on the next three pages where you will have lots of space.

You should include appropriate comments in your code.

Problem 17 (10 points)
If the style of the ~~sixth~~ homework, implement the `postiveSum` function as a C function that only uses two control structures:

```
goto label ;
if (expression) goto label ;
```

Do not use the `?:` operator of C (and Java) to simulate an `if-then-else`.

This specifically means that you can't use the `for`, `while`, `switch`, `break`, `continue`, or even the statement block delimiters `{` and `}`. You can use the `if`, but only when the conditional expression is immediately followed by a `goto` statement.

Problem 17, C with goto, answer goes here

Your answer starts here. Some code has been provided for you.

```
int positiveSum(int V[], int size) {

  int sum = 0 ;
```

```
  return sum ;
}
```