

**UNCA CSCI 235**  
**Exam 2 Spring 2019**  
 29 April 2019s

This is a closed book and closed notes exam. Communication with anyone other than the instructor is not allowed during the exam. **Furthermore, calculators, cell phones, and any other electronic or communication devices may not be used during this exam.** Anyone needing a break during the exam must leave their exam with the instructor. Cell phones or computers may not be used during breaks.

*This exam must be turned in before 6:55 PM.*

Name: \_\_\_\_\_

**Problem 1 (10 points) C expressions**

In the left column, there are ten tricky and not-so tricky C expressions. Write their values in the right column. Express your answers as simple base 10 expressions, such as 235 or -235. You may assume that all of these numbers are stored in 16-bit two's complement representation, the usual short.

<b>0100</b>	<b>64</b>
<b>0x64</b>	<b>100</b>
<b>23 * 4 / 5</b>	<b>18</b>
<b>23 / 4 * 5</b>	<b>25</b>
<b>23 &amp;&amp; 24</b>	<b>1</b>
<b>23 &amp; 24</b>	<b>16</b>
<b>23 &lt;= 3</b>	<b>0</b>
<b>23 &lt;&lt; 3</b>	<b>184</b>
<b>~23</b>	<b>-24</b>
<b>23 * 235 &amp;&amp; 4 - 4</b>	<b>0</b>

**Problem 2 (4 points) Q4.4 to decimal conversion**

Convert the following two Q4.4 *two's complement* numbers (four fixed and four fractional bits) into conventional decimal numbers.

<b>11010100</b> <b>-2.75</b>	<b>00101010</b> <b>2.625</b>
---------------------------------	---------------------------------

**Problem 3 (4 points) Decimal to Q4.4 conversion**

Convert the following two signed decimal numbers into Q4.4 *two's complement* numbers (four fixed and four fractional bits). If you can't express the number exactly, give the nearest Q4.4 representation.

<p>-4.25 <b>10111100</b></p>	<p>3.166666666 <b>00110011</b></p>
----------------------------------	--

**Problem 4 (10 points) Adding numbers with flags**

Add the following pairs of six-bit numbers. Based on the result of this addition, set the four x86-64 status bits: CF (carry), OF (overflow), SF (sign) and ZF (zero).

<p><b>010001</b> + <b>001101</b> <b>011110</b> CF <b>0</b>, OF <b>0</b>, SF <b>0</b>, ZF <b>0</b></p>	<p><b>110110</b> + <b>001010</b> <b>000000</b> CF <b>1</b>, OF <b>0</b>, SF <b>0</b>, ZF <b>1</b></p>
<p><b>011110</b> + <b>011110</b> <b>111100</b> CF <b>0</b>, OF <b>1</b>, SF <b>1</b>, ZF <b>0</b></p>	<p><b>101000</b> + <b>101000</b> <b>010000</b> CF <b>1</b>, OF <b>1</b>, SF <b>0</b>, ZF <b>0</b></p>

**Problem 5 (4 points) Range**

What is the range of numbers that can be stored in 8-bit twos-complement numbers? (The byte of Java is an 8-bit twos-complement number.)

**-128 to 127**

**Problem 6 (4 point) Range**

What is the range of numbers that can be stored in 16-bit unsigned numbers? (The char of Java is a 16-bit unsigned number.)

**0 to 65535**

### Problem 7 (6 points) CSCI arithmetic

Perform the following operations and express the results as they should be for CSCI 235 and other geeky environments. *Use powers of 2!*

$$32 \text{ ki} * 32 \text{ ki} = 2^5 * 2^{10} * 2^5 * 2^{10} = 2^{30} = 1 \text{ Gi}$$

$$4 \text{ ki} / 64 = 2^2 * 2^{10} / 2^6 = 2^6 = 64$$

$$\log_2(4 \text{ Gi}) = \log_2(2^2 * 2^{30}) = \log_2(2^{32}) = 32$$

### Problem 8 (6 points) Compile and run

Suppose you have written the following program and stored it in a file named `count.c`.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("%s %s %d\n", argv[0], argv[1], argc) ;
}
```

What would be an appropriate **Makefile** for the program? Assume you want to store the executable in the file `count`. I am giving you the first line of the **Makefile**. You need to put in the others. Write your **Makefile** so that typing the command `make` (with no arguments) will compile your program.

```
CFLAGS = -std=c11 -Wpedantic -Og -g
count: count.c
```

Now comes the tricky part of the riddle. Below is the start of a single command typed from the command prompt. What command line arguments do you pass to `count` to make it print this single output line?

```
count to 3
```

Just complete the command.

```
count to 3
no kidding...
```

PS: `args` in Java is like the combination of `argv` and `argc` in C.

### Problem 9 (20 points) C Programming

Write a program that reads (**using scanf**) a bunch of course “numbers” (a string, four characters or less, followed by a number, 999 or less) from a terminated standard input stream that are entered as shown below.

```
        CSCI 313   ACCT
215   JDB  19     NM 313   MATH 251
```

Your output should be a neatly formatted list of courses (one per line as shown below) followed by a count of the by the number of courses. (Assume all courses appear only once.) So, for the above example, the output should something like

```
CSCI  313
ACCT  215
JDB   19
NM    313
```

```
course count:      4
```

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
    char courseName[5] ;
    int courseNumber ;
    int courseCount = 0 ;
    while(scanf("%4s %d", courseName, &courseNumber) == 2) {
        ++courseCount ;
        printf("%-4s %6d\n", courseName, courseNumber) ;
    }
    printf("course count: %6d\n", courseCount) ;
}
```

**Problem 10 (10 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 Java (and C) expression:

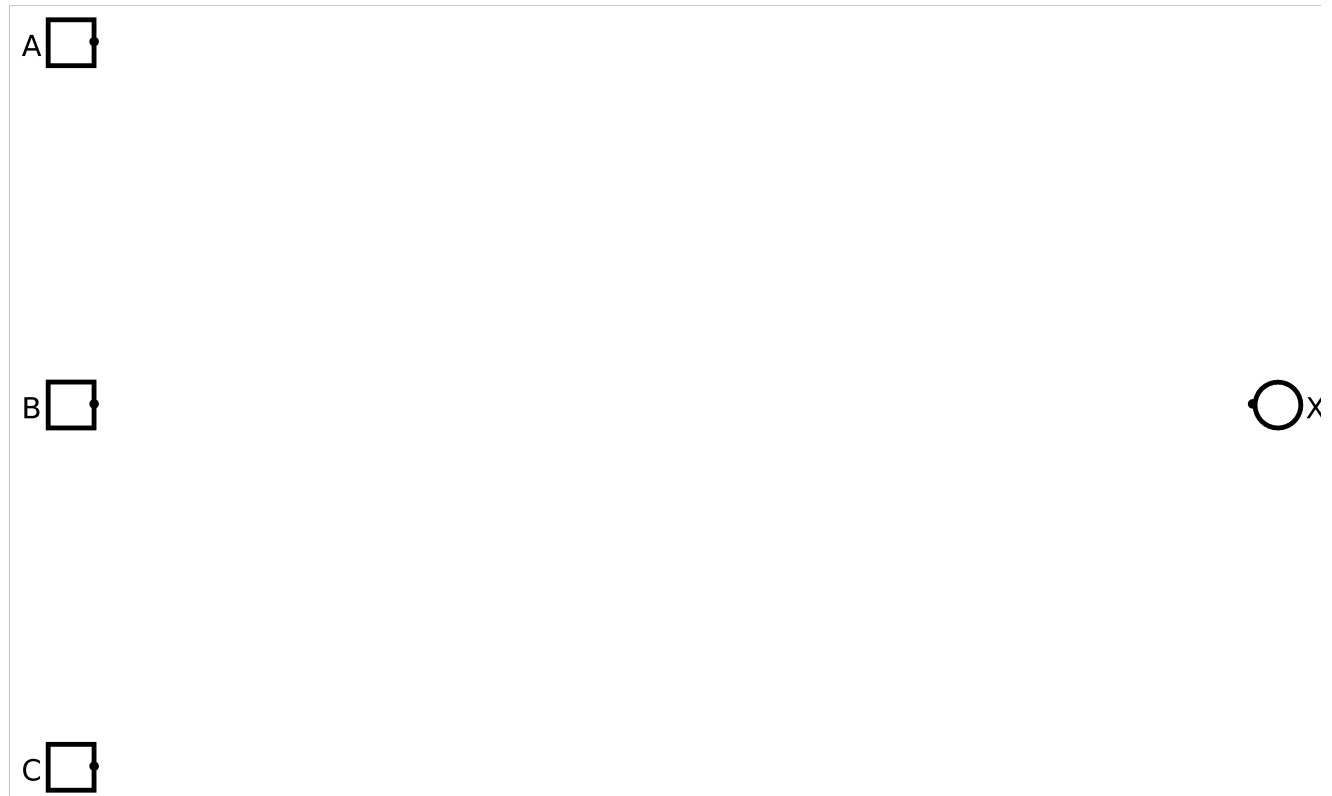
$$X = !(A \ || \ B) \ \&\& \ C$$

If you prefer the computer engineering style, you can think of the equation as

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

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

**Second**, draw a logic circuit (AND, OR, ...) to implement the boolean expression and corresponding truth table.



**Problem 11 (10 points) Truth table to Boolean expression and circuit**

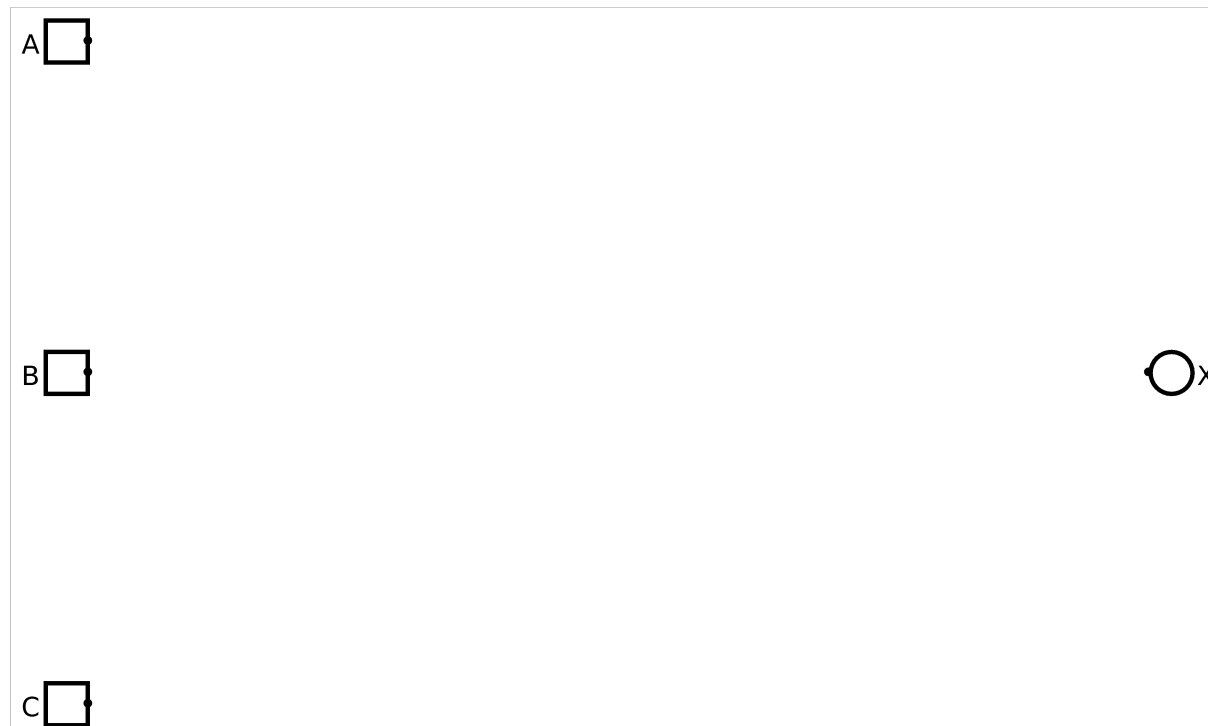
The truth table below specifies a Boolean function with three inputs, **A**, **B**, and **C** and one output **X**.

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

**First**, write a Boolean expression corresponding to the function specified in the table. You do not need to write an “efficient” expression; however, ridiculously complex expressions will not be given full credit.

$A' B' C + A' B C' + A B' C + A B C$  or  $B' C + A' B C' + A C$

**Second**, draw a logic circuit (AND, OR, ...) to implement the boolean expression and corresponding truth table.



### Problem 12 (12 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    V[3] = {201, 235, 335} ;
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.

```
p = &V[0] ;
*p = 201 ;
p = p + 2 ;
*p = 202 ;
```

p

q

V[0]	201
V[1]	235
V[2]	202

```
q = V ;
p = q + 2 ;
*p = *q + 2 ;
```

p

q

V[0]	201
V[1]	235
V[2]	203

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

p

q

V[0]	201
V[1]	100
V[2]	1

```
p = &V[0] ;
*p++ = 1000 ;
*++p = 2000 ;
```

p

q

V[0]	1000
V[1]	235
V[2]	2000

# CSCI 255

## Handy Table of Numbers

### Powers of Two

$2^0$	1
$2^1$	2
$2^2$	4
$2^3$	8
$2^4$	16
$2^5$	32
$2^6$	64
$2^7$	128
$2^8$	256
$2^9$	512

$2^{10}$	1024
$2^{11}$	2048
$2^{12}$	4096
$2^{13}$	8192
$2^{14}$	16384
$2^{15}$	32768
$2^{16}$	65536
$2^{17}$	131072
$2^{18}$	262144
$2^{19}$	524288

$2^{10}$	1 Ki
$2^{20}$	1 Mi
$2^{30}$	1 Gi

### Hex table

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111