**UNCA CSCI 235**
**Final Exam Fall 2018**
11 December 2018 – 3:00 pm to 5:30 pm

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.

Name:______________________________

**Problem 1 (10 points) C expressions**
In the left column, there are fifteen 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`.

| | |
|---|---|
| `0123` | 83 |
| `0xEB` | 235 |
| `14 && 14` | 1 |
| `42 & 35` | 34 |
| `42 >> 3` | 5 |
| `42 \| 35` | 43 |
| `42 << 3` | 336 |
| `42 ^ 35` | 9 |
| `~42 + 1` | -42 |
| `3 * 4 / 5` | 2 |
| `(3 * 4) / 5` | 2 |
| `1 < (2 < 3)` | 0 |
| `17 & ~17` | 0 |
| `17 && -17` | 1 |
| `(17 == 17) * 17` | 17 |

**Problem 2 (4 points) Decimal to two's complement conversion**
Convert the following four signed decimal numbers into **five**-bit *two's complement* representation. Some of these numbers may be outside the range of representation for **five**-bit two's complement numbers. Write "out-of-range" for those cases.

| | |
|---|---|
| **15**<br><br>**01111** | **32**<br><br>***out-of-range*** |
| **-15**<br><br>**10001** | **-32**<br><br>***out-of-range*** |

**Problem 3 (3 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.

| | |
|---|---|
| **11001000**<br><br>**-3.5** | **00010011**<br><br>**1.1875** |

**Problem 4 (3 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.

| | |
|---|---|
| 3.3<br>**00110101** | -1.414<br>**11101001**<br>**-2 + 0.586 ~~ -2 + 9/16** |

**Problem 5 (6 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).

| | |
|---|---|
| 111011<br>+ 000101<br>000000<br>CF_1, OF_0, SF_0, ZF_1 | 011100<br>+ 000100<br>100000<br>CF_0, OF_1, SF_1, ZF_0 |
| 100000<br>+ 100000<br>000000<br>CF_1, OF_1, SF_0, ZF_1 | 010110<br>+ 000110<br>011100<br>CF_0, OF_0, SF_0, ZF_1 |

**Problem 6 (2 points) Range**
What is the range of numbers that can be stored in 16-bit twos-complement numbers? (The `short` of Java is a 16-bit twos-complement number.)

**-32768 to -32767**

**$2^{15}$ to $2^{15}$-1**

**Problem 7 (2 points) 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 65536**

**0 to $2^{16}$-1**

**Problem 8 (6 points) Formatted printing**

Suppose that the `int` variable C has the value 170 (in decimal). The left column in the table below has a `printf` statement. The right column has the desired output for that `printf` within a six character field. Your task is to fill in the underlined part (the stuff after the %). ***You must use a single "conversation specifier" (the thing starting with a %) in your format string. No "ordinary characters" are allowed.*** This means the following are not allowed because they contain ordinary characters.

```
printf("000160", C) ;   // contains only ordinary characters
printf("   %3d", C) ;   // starts with three ordinary characters
```

| | |
|---|---|
| `printf("%6d",C) ;` | _ _ _ 1 6 0 |
| `printf("%06d",C) ;` | 0 0 0 1 6 0 |
| `printf("%+6d",C) ;` | _ _ + 1 6 0 |
| `printf("%6o",C) ;` | _ _ _ 2 4 0 |
| `printf("%6x",C) ;` | _ _ _ _ a 0 |
| `printf("%6X",C) ;` | _ _ _ _ A 0 |

**Problem 9: goto programming (8 points)**
In the style of a recent homework, implement the C function shown below using only two control structures:

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

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

```
int population(unsigned int p) {
    int c = 0 ;
    while (p > 0) {
        if (p & 1) {
           ++c ;
        }
        p = p >> 1 ;
    }
    return c ;
}
```

```
int population(unsigned int p) {
    int c = 0 ;
    goto loopTest ;
LoopStart:
    if (!(p & 1)) goto skipIncrement ;
    ++c ;
SkipIncrement:
    p = p >> 1 ;
loopTest:
    If (p > 0) goto loopStart ;
  return c ;
}
```

**Problem 10 (6 points) & and >>**
The following expressions and declarations were used in the preceding question to determine the "population" (number of 1's) in an integer. Answer a couple of questions about this census program.

What is the role of the expression "`p & 1`" and the statement "`p = p >> 1`" in calculating the number of 1's?

**p & 1 tests if the rightmost bit is a 1.**
**p = p >> 1 moves the bits right one position.**

Why must the parameter `p` be `unsigned`?

**If p is unsigned, the bit moved into the leftmost position by p = p>>1 is guaranteed to be a 0.**

**Problem 11 (5 points) CSCI arithmetic**
Perform the following operations and express the results as they should be for CSCI 235 and other geeky environments. *You* ***must*** *use powers of 2!*

| |
|---|
| **4 Mi * 32 ki = $2^2 * 2^{20} * 2^5 * 2^{10}$ = $2^{37}$ = 128 Gi** |
| **16 Mi / 64 = $(2^4 * 2^{20}) / 2^6$ = $2^{18}$ = 256 ki** |
| **$\log_2$(16 ki) = $\log_2(2^4 * 2^{10})$ = $\log_2(2^{14})$ = 14** |

**Problem 13 (5 points) C Programming**
Write a program that reads (**using scanf**) letters (from A to Z). Such as

```
A  B  C  C  A
```

Your program should sum the number of time each letter appears in a neat table such as:

```
A:   2
B:   1
C:   2
……
Z:   0
```

```
#include <stdio.h>
int main(int argc, char *argv[]) {
  int numLetters[26] ;
  char nextLetter ;
  for (int i = 0; i<26; ++i)
    numLetters[i] = 0 ;
  while (scanf("%c", &nextLetter) == 1)
    numLetters[nextLetter-'A']++ ;
  for (int i = 0; i<26; ++i)
    printf("%c:%6d\n", i+'A', numLetters[i]) ;
}
```

*You can do the same array magic in Java!*

```
import java.util.Scanner;
public class prob13 {
  public static void main(String[] args) throws java.io.IOException {
    int numLetters[] = new int[26] ;
    int nextLetter ;
    while ((nextLetter = System.in.read()) != -1) {
      if ('A' <= nextLetter && nextLetter <= 'Z')
        numLetters[nextLetter-'A']++ ;
    }
    for (int i = 0; i<26; ++i) {
      System.out.format("%c:%6d\n", i+'A', numLetters[i]) ;
    }
  }
}
```

**Problem 13 (5 points) Boolean expression to truth table**

Fill in the truth table on the right below so that it corresponds to the following Java (and C) expression:

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

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

```
X = (A' + B C) C'
```

| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

**Problem 14 (5 points) Truth table to Boolean expression**

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

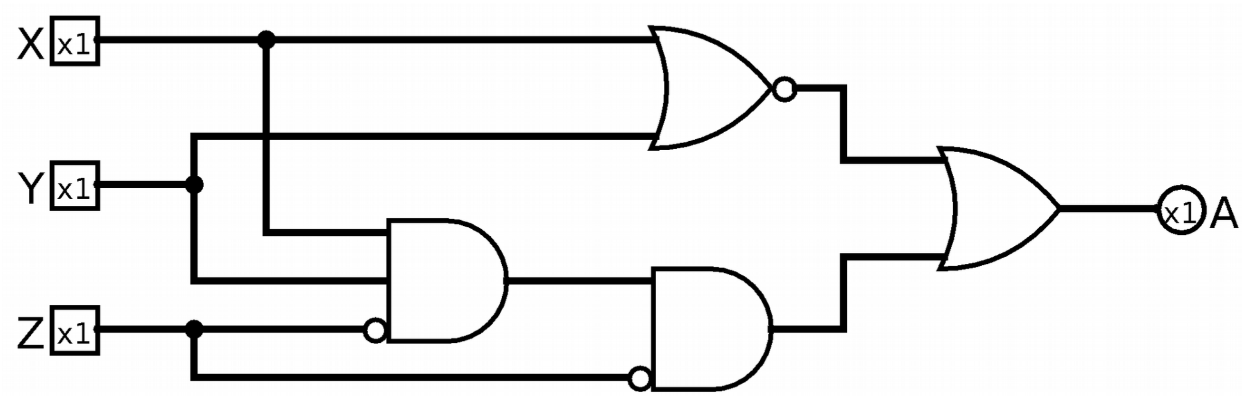
**A' B' C'**
**+**
**A' B C**
**+**
**A B C'**

| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

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. The phrase "ridiculously complex expressions" means "expressions with require more than five minutes of instructor time to decode".

*Do it mechanically. Avoid cleverness.*

**Problem 15 (8 points) Circuit to Boolean expression and truth table**
A gate-level circuit is shown below with three inputs on the left and a single output on the right.




First, write the Boolean expression corresponding to this circuit. (Don't worry about the "x1". It indicates that the connection is for a single bit.)

**(X + Y)' + (X  Y Z') Z'**
**(X + Y)' is 1 only if both X and Y are 0**
**(X Y Z') is 1 only when X and Y are 1 and Z is 0**

Next, complete the following truth table so that it corresponds to this digital logic circuit.

| X | Y | Z | A |
|---|---|---|---|
| 0 | 0 | 0 | **1** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **0** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **0** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | **0** |

*An ugly example. It is really much too complicated. Few people aced it.*

**Problem 16: Definitions (7 points)**
Give short definitions of the following concepts, functions, hacks, programs, types, variables, etc., you have seen in the labs and homework of this course, *Feel free to skip one: I will grade the best seven of eight definitions.*

| |
|---|
| `-std=c99` |
| `-Wpedantic` |
| `analogWrite()` |
| Circuit Playground Express |
| `logisim` |
| `nano` |
| `opendir()` |
| `qsort()` |

**Problem 17 (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    V[3] = {201, 235, 335} ;
int    *p = NULL ;
int    *q = NULL ;
```

As you know, **`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] ;
q = &V[2] ;
*p = *q ;
++*p ;
```

| | | | |
|---|---|---|---|
| p | &V[0] | V[0] | 336 |
| | | V[1] | 235 |
| q | &V[2] | V[2] | 335 |

```
q = V ;
p = q + 1 ;
*p = 181 ;
*q = *p + 100 ;
```

| | | | |
|---|---|---|---|
| p | &V[1] | V[0] | 281 |
| | | V[1] | 181 |
| q | &V[0] | V[2] | 335 |

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

| | | | |
|---|---|---|---|
| p | &V[1] | V[0] | 201 |
| | | V[1] | -234 |
| q | &V[2] | V[2] | 1 |

```
p = &V[0] ;
q = &V[2] ;
*q = (*p)++ ;
```

| | | | |
|---|---|---|---|
| p | &V[0] | V[0] | 202 |
| | | V[1] | 235 |
| q | &V[2] | V[2] | 201 |

*These were checked with some weird C code*

# 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 |