CMSC 14300 - Summer 2025 — Konstantinos Ameranis

Final Review

Quiz time

The quiz will be held on **Monday 7/14/2025 from 6-8pm** in **Ryerson 251**. The exam period will be **120 minutes** long. You can bring one hand-written (not printed) sheet as reference. Additionally, you can have a hand held, non programmable calculator. All other resources, including books, printed notes, electronic access or communication is not allowed and can result in a zero on the exam.

Here is a list of topics:

1. Variables
   - `int`
   - signed vs unsigned
   - `char`
   - `float`
2. Strings
   - Working with strings
   - Reading, printing and manipulation
3. Converting to different bases
   - Decimal
   - Binary
   - Hexadecimal
   - Octal
4. Floating point arithmetic
   - Exponent
   - Mantissa
   - Being able to translate a floating point to decimal
5. Bit packing
   - Masking
   - Setting bits
   - Retrieving information from binary

    - Resetting fields

6. Command line arguments
7. Pointers
    - Dereference operator `*`
    - Address operator `&`
    - Pointer arithmetic
    - Deep vs shallow copy
8. Memory Management
    - Stack vs heap
    - Automatic vs manual management
    - Memory leaks

Problem 1 (Bits (8 points)). 1. (2 points) What is the smallest `int8_t` number?

2. (2 points) What is the largest positive subnormal `float`?

3. (2 points) Write `10010011`, which is a *signed* 8-bit binary number, in decimal

4. (2 points) Write `10010011`, which is an *unsigned* 8-bit binary number, in decimal

Problem 2 (Mystery (18 points)). You just started working for a company that is implementing a set of procedures to operate on bits, but the code base is full of undocumented C code with magic numbers and bitwise operations aplenty. To do anything, you must first understand what the functions are currently doing.

First, you discover the following function, which takes a 16-bit signed integer as input and returns a 16-bit signed integer. *Hint: ^ is the XOR operator.*

```
int8_t mystery1(int8_t n)
{
    int8_t m = n >> 7;
    return (n + m) ^ m;
}
```

(a) (5 points) For each of the following values of n, calculate the intermediate values.

n	n	m	(n + m)^ m	(n + m)^ m
(Decimal)	(Binary)	(Binary)	(Binary)	(Decimal)
13				
-7				

(b) (3 + 2 points) Briefly explain in plain language what mystery1 does, and explain why this function works. *(You will not receive any credit for merely translating each line in words — "n is shifted 31 bits to the right, followed by an addition..." is not acceptable.)*

Then, you encounter the following function, `mystery2`.

```
uint32_t mystery2(uint32_t n, int i, uint8_t b)
{
    uint32_t m = 0xFF << (i << 3);
    n &= ~m;
    n |= b << (i << 3);
    return n;
}
```

(c) (3 points) For each of the following lists of arguments, calculate what `mystery2` returns.

n (Hexadecimal)	i (Decimal)	b (Hexadecimal)	mystery2(n, i, b) (Hexadecimal)
0x1A2B3C4D	0	0xFF	
0x1A2B3C4D	2	0xAB	

(d) (5 points) What does `mystery2` do?

Problem 3 (Bit-packing (15 points)). Consider the following 8-bit encoding of a student information, from the *highest*[1] bit to the *lowest* bit:

- A 2-bit unsigned number for the student's year (0 for first year, up to 3 for fourth year) in the program;
- A bit indicates whether a student is a transfer student (1 for transfer student);
- A bit indicates whether a student lives on campus (1 for on-campus); *and*
- A 4-bit unsigned number indicates the student's birth month (0 for January, up to 11 for December).

(4 points) Write an 8-bit number in binary characterizing the following student a third-year, non-transfer student, living on campus, born in July.

(6 points) Write a C function that takes an 8-bit number in the encoded student information described above, and returns non-zero if and only if the student is a *transfer* student who lives *on campus*.

```
int is_transfer_on_campus(uint8_t student){











}
```

[1] Corresponding to 2^7

(i) (5 points) The number of 1-bits in a binary number is also known as the *population* of the number. For example, the population of the number 7 (`0000 0111`) is 3. Write a C function that computes the population of a given 64-bit number.

```
int popcount(uint64_t n){











}
```

Problem 4 (Writing C (10 points)). (10 points) Write a function that takes a C string and returns a fresh heap-allocated string with a period inserted *after* every character. For example, if the string given is "July", the return value should be string "J.u.l.y.". Note that there is no restrictions on how long the input string is. You can assume that the argument is not NULL.

```
char* dots(char* str){
















}
```