Background

In APP C25-1 EXT, you practiced converting numbers from decimal (base 10) to octal (base 8) and vice versa. In this EXT assignment, you will use that knowledge to implement **bitwise AND** and **bitwise OR operators** for 3-digit octal numbers. Bitwise operations are useful because they can be combined to perform calculations usually much faster and with fewer resources than addition, multiplication, and division.

To do this, you will have to convert octal (base 8) numbers into **binary** (base 2) numbers. The digits for binary numbers are called **bits**, and these bits can only be 0 or 1. There is a special relation between octal numbers and binary numbers which makes it easy to convert between them. Each octal digit independently represents exactly 3 binary bits. This is because the maximum octal digit value of 7 can be represented with just three binary digits (all 1 bits). Thus, you can apply the same conversion algorithm from APP C25-1 EXT to convert each octal digit into 3 binary bits. An example of this is shown below for a 3-digit octal number converting into a 9-bit binary number.

Convert octal (base 8) to binary (base 2)

Number = 472 (base 8)

Octal:	4	7	2
	4 / 2 = 2 remainder 0 ↑	7 / 2 = 3 remainder 1 ↑	2 / 2 = 1 remainder 0 ↑
	2 / 2 = 1 remainder 0	3 / 2 = 1 remainder 1	1 / 2 = 0 remainder 1
	1 / 2 = 0 remainder 1	1 / 2 = 0 remainder 1	0 / 2 = 0 remainder 0
Binary:	100	111	010

→ Number = 100111010 (base 2)

In digital logic, the general AND and OR operations can be applied to a pair of bits according to the tables below. The output of the AND operation is 1 if both of the input bits are 1, and 0 otherwise. The output of the OR operation is 1 if either (or both) of the input bits are 1, and 0 otherwise.

AND		
Bit 1	**Bit 2**	**Result**
0	0	0
0	1	0
1	0	0
1	1	1

OR		
Bit 1	**Bit 2**	**Result**
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise AND and OR operators take two binary (base 2) numbers and apply either the AND operation or the OR operation to each pair of bits across the two numbers, and the individual bit results are put together in the proper order to form a new binary (base 2) number. Two examples are shown on the next page for a 4-bit representation.

Bitwise AND	**Bitwise OR**
Number 1 = 1100 (base 2) Number 2 = 1010 (base 2) 1100 AND 1010 --------------- 1000 Number 3 = Number 1 bitwise AND Number 2 = 1000 (base 2)	Number 1 = 1100 (base 2) Number 2 = 1010 (base 2) 1100 OR 1010 --------------- 1110 Number 3 = Number 1 bitwise OR Number 2 = 1110 (base 2)

The above examples are for a 4-bit representation. Note that for 3-digit octal numbers, the corresponding binary numbers have 9 bits. The procedure above is the exact same for 9-bit bitwise operations. For an unsigned n-bit representation, the smallest decimal (base 10) number which can be represented is 0 (corresponding to all 0 bits), and the largest decimal (base 10) number which can be represented is $2^n - 1$ (corresponding to all 1 bits).

Instructions

- Complete the C26-1 assignment which introduces arrays in C.
- Create a new program.
- Prompt the user to enter to enter two 3-digit octal (base 8) numbers. Save each octal number into an array of integer digits.
  - You may choose to have the user enter each digit individually to make the scanning easier.
  - Think about in what order the user should enter the digits and in what order you will save them into the array to make further operations easier later. You may want to talk with others at your table and work it out together on a whiteboard.
- Convert the two entered 3-digit octal numbers into 9-bit binary numbers. Save each conversion result into an array of integer digits.
  - Think carefully about how to conserve the proper order of digits and bits based on how you scanned in the user input. The final order of binary bits printed to the screen matters.
  - This is probably the most difficult part of the assignment and will test your ability to manipulate arrays.
- Display the binary conversion results for both numbers to the screen.
- Perform the bitwise AND operation on the two binary numbers. Save the resulting 9-bit binary number in an array of integer digits.
  - You may use the `&&` operator but you MAY NOT use `&` by itself.
- Perform the bitwise OR operation on the two binary numbers. Save the resulting 9-bit binary number in an array of integer digits.
  - You may use the `||` operator but you MAY NOT use `|` by itself.
- Display the bitwise AND and bitwise OR operation results to the screen.