Part I: Bitwise operators and Bit-masking

C provides bitwise commands for AND(&), OR(|), XOR(^), and NOT(^). Ignoring NOT for now, let’s see what happens when we reduce the 2-input gates to 1-input gates by fixing the second input.

a. Let x be the input. Fill in the following blanks with either 0, 1, x, or \(\overline{x}\) (NOT x):

\[
\begin{align*}
    x \& 0 &= \_\_ \\
    x | 0 &= \_\_ \\
    x ^ 0 &= \_\_ \\
    x \& 1 &= \_\_ \\
    x | 1 &= \_\_ \\
    x ^ 1 &= \_\_ 
\end{align*}
\]

b. Based on your responses, look at the columns (grouped by operation) above. Which operation would be useful for turning bits OFF? For turning bits ON? For flipping bits?

From K&R: Write a function, \(\text{invert}(x, p, n)\), that returns \(x\) with the \(n\) bits that begin at position \(p\) inverted, leaving the others unchanged.

Part II: C and Pointers

Write the following functions so that they perform according to the following descriptions. Not all are guaranteed to be soluble.
1. Write a program that prints the string "Hello World" to standard output. For this question only, write the entire contents of the C file and try to make your solution stylistically ideal.

2. Swaps the value of two ints outside of this function.

3. Increments the value of an int outside of this function by one.

4. Returns the number of bytes in a string. Do not use strlen.

5. Returns the number of elements in an array ARR of ints. The array must be able to store any integer that fits in the int type.