Lab6
Deadline: In lab in the week of Mar 18

Requirements
The sat program performs a synthetic saturating multiplication operation. Saturating multiplication clamps the result into the representable range. Instead of overflowing with wraparound as ordinary two's-complement addition does, a saturating addition returns the type's maximum value when there would be positive overflow, and minimum when there would be negative overflow. Saturating arithmetic is a common feature in 3D graphics and digital signal processing applications.

Here are two sample runs of the sat program:
```
>>cat input.txt
8
E

>>./a.out input.txt output.txt
>>cat output.txt
min: -128	0xffffffffffffff80
max: 127	0x000000000000007f

>>cat input.txt
8
1
5
E

>>./a.out input.txt output.txt
>>cat output.txt
5

>>cat input.txt
8
126
-5
E

>>./a.out input.txt output.txt
>>cat output.txt
-128
```

The E indicates end of file in input.txt file. When displaying the min/max value, you must display the hexadecimal value with zero padding. The delimiter between the min/max signed value and the hexadecimal must be \t. Everything else must be a single space. The example program above reports that an 8-bit signed value has a range of -128 to 127 and if you attempt to multiply 126 by -5, the result underflows and sticks at the maximum value of -128.

Restrictions

- The bit width is a number between 4 and 64. A two's-complement signed value is used in this lab.
- No relational operators or math.h functions. You are prohibited from making any use of the relational operators. This means no use of < > <= >=. You may use != == for logical comparisons. You also should not call any function from the floating point math.h library (e.g no pow, no exp2). These restrictions are intended to guide you to implement the operation via bitwise manipulation. All other operators (arithmetic, logical, bitwise, ...) are fine.
- No special cases based on bitwidth. Whether the value of bitwidth is 4, 64, or something in between, your functions must use one unified code path to handle any/all values of bitwidth without special-case handling. You should not use if/switch/? to divide the code into different cases based on the value of bitwidth. This doesn't mean that you can't use conditional logic (such as to separately handle overflow or non-overflow cases), but conditionals that dispatch based on the value of bitwidth or make a special case out of one or more bitwidths are disallowed.
- There should be no extra trailing line in the output.txt file.
- A solution that violates any of these restrictions will receive zero, so please verify your approach is in compliance.

How to Compile and Run

- The Makefile for lab is provided.
- The Makefile is supposed to work with lab6.c, input.txt, output.txt and ref.txt files so, make sure to name your files accordingly.
- Run the following command in vs code Terminal.
  ```
  make
  ```
  It should compile the code without any errors.
  ```
  make convert_input
  ```
  It should convert the input.txt file to unix encoding.
  ```
  make run
  ```
  It should run the compiled code.
- Run the following command to delete the out file.
  ```
  make clean
  ```
- Run the following command to convert the generated output to unix encoding.
  ```
  make convert_output
  ```
  It should convert the output.txt file to unix encoding.
- Run the following command to check your output with provided ref file.
  ```
  make check
  ```

- You are not supposed to make any changes in the Makefile.
- Make sure to install dos2unix utility using the following command:

```
sudo apt-get install dos2unix
```

  For Mac

```
brew install dos2unix
```

Grading

Any grading failure due to not following specifications will result in 0. For full marks this week, you must:

- (1 point) Correctly submit A number file
- (4 point) Generate a correct solution to the problem(s) in this lab

Submission Files

- You must deliver only one .c file named: **lab6.c** (do not capitalize)