**Lab06**
**Deadline: 11:59PM Mar 10**

**Requirements**

The `sat` program performs a synthetic saturating addition operation. Saturating addition 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:

```
>>./a.out 8
8-bit signed integer range
min: -128   0xffffffffffffff80
max: 127    0x000000000000007f
>>./a.out 8 126 5 0
126 + 5 = 127
>>./a.out 8 126 -5 0
126 + -5 = 121
>>./a.out 8 -126 5 0
-126 + 5 = -121
>>./a.out 8 -80 5 1
-80 - 5 = -85
```

When displaying the min/max value, you must display the hexadecimal value. The delimiter between the min/max signed value and the hexadecimal must be `\t`. Everything else must be a space. Make sure that no extra spaces or newlines before and after. The above code is a good example of what needs to be printed in your console without any trailing spaces. Note that there is a space between `126` and `+` as well. The example program above reports that an 8-bit signed value has a range of -128 to 127 and if you attempt to add 126 to 5, the result overflows and sticks at the maximum value of 127. The last number represents whether it is an addition or a subtraction. 0 represents addition while 1 represents subtraction. Your task is to implement the functions below to support saturating addition for the sat program.

```
long long signed_min(int bitwidth);
long long signed_max(int bitwidth);
long long sat_add(long long operand1, long long operand2, int
bitwidth);
long long sat_sub(long long operand1, long long operand2, int
bitwidth);
```

The `bitwidth` argument is a number between 4 and 64. A two's-complement signed value with `bitwidth` total bits is limited to a fixed range. The `signed_min` and `signed_max` functions return the smallest and largest values of that range. The `sat_add/sat_sub` function implements a saturating addition/subtraction operation which returns the sum/difference of its

operands if the sum/difference is within range, or the appropriate min/max value when the result overflows. The type of the two operands is `long long` but you can assume the value of the operands will always be within the representable range for a bitwidth-sized signed value. That being said, this means there may be more bits than you need for a bitwidth-sized value, and these extra bits should be set appropriately to ensure the value is correctly interpreted when it is inside a long long.

**Restrictions**

- 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 != ==. 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.
- A solution that violates any of these restrictions will receive zero, so please verify your approach is in compliance.
- There will be no corner case testing.

**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
- (1 point) Not having any files in github other than lab6.c and AXXXX.txt
- (2 point) Generate a correct solution (including correct memory allocation and deallocation) to the problem(s) in this lab

**Submission Files**

- You must deliver only one .c file named: **lab6.c** (do not capitalize)
- AXXXX.txt (empty file, but with your A number as file name. Make sure to include 0's, match this A number with your A number in learning hub, and have .txt extension)
- Github: https://classroom.github.com/a/4m60ucUd