In this lab you will gain more experience in manipulating different data types in C, you will revisit the modulus operator (that you have learned in previous CS courses), and you'll perform a series of bit manipulations. You'll write a main program that prompts the user for inputs, and which then invokes the functions that you'll write.

I. Modulus by Brute Force to Convert to a Different Base

The Objectives of this part of the lab are the following:

- Become comfortable with using a char pointer
- Perform modular arithmetic
- Continue to develop skills at writing C functions

As was the case for lab 1, ask the TAs lots of questions. They are there to help. And, **you CAN work with peers in completing the labs. This is a bit tricky when everybody is remote; you may use slack, email, discord, etc. Up to you.** But, in no circumstances should you SHARE code, which involves emailing code and then submitting it as your own.

Submitting your work

Submit your C program files as the Lab 2 Submission item on Canvas. You must submit your program by the due date/time specified on Canvas.

Create a C source file, compile it, and link it to get an executable

Use the nano editor (or any editor of you choice, such as emacs) to create a file called *lab2Convert.c*, type the following contents and save the file.

```
#include <stdio.h>
int main() {



}
```

Compile this file into an executable by issuing the following from the command line, and then run:

```
$ gcc -o lab2Convert lab2Convert.c
```

```
$ ./lab2Convert
```

This is a skeleton of a program, so it doesn't do anything (yet).

Implement a Conversion Function

Implement a function with the following signature:

```
char* itoa(int num, char* str, int base);
```

where `num` is an integer that you need to represent as a string in the `base` specified. The input can be a negative number. In principle, this is VERY similar to the task for homework 1, but here you are tasked to write the function for converting to ANY base, and not just binary nor hexadecimal.

For example, assuming `char buffer[50];` has been declared,

Invocation	Output to screen
`printf("%s",itoa(4, buffer, 2));`	`100`
`printf("%s",itoa(64, buffer, 8));`	`100`
`printf("%s",itoa(72, buffer, 8));`	`110`
`printf("%s",itoa(675, buffer, 16));`	`2A3`
`printf("%s",itoa(456, buffer, 12));`	`320`

Hint: To specify the alphabet of a base greater than 10, use the ASCII values of letters starting with A, then B, the C, the D, etc.

A typical approach is the following:

1. Get the least significant digit by finding the modulus of the number and the base
2. Depending on the base, represent this digit in ASCII by either adding it to the ASCII ‘a’ or the ASCII ‘0’ and store it in a string
3. Divide the number by the base and repeat steps 1-3 until the number is reduced to zero
4. Reverse the string (the `char* str`), and display using your method of choice.

II. Bit manipulation

The Objectives of this part of the lab are the following:

- Continue to develop your understanding of binary representation of integers

Implement a function with the following signature:

```
int countSetBits(unsigned int var);
```

This function should return the number of set bits (`1`s) in argument. That means, you must convert the input unsigned int into its binary equivalent, and then count the number of bits. Note: This is in principle the same as the bitCount function of project 1, but here you ARE allowed to use any operator that you want. Also, this function's input is an unsigned int, which is a bit different than the bitCount function in project 1.

Implement a function with the following signature:

```
int reverseBits(int var);
```

This function should return an integer that has all the bits (including the sign bit) of the argument reversed. These functions that perform bit manipulations are meant to force you to THINK about and MANIPULATE the bit representations of different data types. For example, there are many learning points for this task. First you would need to recall that ints in C are 4 bytes and are SIGNED. Thus, invoking `reverseBits(1)` where the 1 is of type int, the 1 would be saved as

`00000 … 00001`

Reversing all of those gives

`10000 … 00000`

Which when assigned to an integer (because that is what the function returns), that integer would have the value

$-1(2^{31}) + 0 + 0 + \ldots + 0$

Thus printing the value of the output of `reverseBits` should print -2147483648 to the screen.

Implement a function with the following signature:

```
bool onlyOneBitSet(int aVar);
```

This function should return *true* if only one bit is set in the argument passed in, and should return *false* otherwise.

III. The main()

Write a main method that

- Prompts the user to provide two integers
- Invokes `itoa` using the two inputs
- Invokes `countSetBits`, `reverseBits`, and `onlyOneBitSet`, using the first input integer
- Prints to the screen the output (char*, int, int, bool) of the 4 functions.

Rubric and submission

Please upload your .c file to Canvas.

`itoa correctly implemented`	3 points
`countSetBits correctly implemented`	3 points
`reverseBits correctly implemented`	3 points
`onlyOneBitSet correctly implemented`	3 points
lab2Convert.c uploaded via Canvas • Program Compiles • Main method prompts for user's input • The 4 functions are invoked • Output is nicely formatted	3 points
	15 points