CSE 1384 - Stacks and Queues
Lab 8

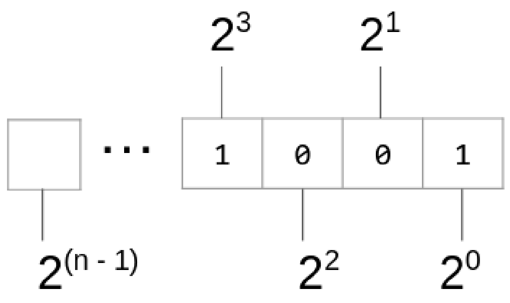
Objectives:

- Continue practicing past concepts
- Practice using stacks and queues in C++

Background:

This week's lab will involve binary to decimal conversion. As such, here's some background/explanation as to how that works if you haven't worked with it before.

Assume you have a four digit binary number – starting from the RIGHT-most digit, this will start at 2^0 and work your way up from 0, for every consecutive digit. So, four digits will have a LEFT-most digit that represents 2^3. This adjusts for every size, where your left-most digit ends up representing 2 brought to the power of the size - 1. A diagram of this concept is found below.



Now, for conversion, you're adding the coinciding 2 exponential only IF the digit in that position is 1. For a couple of examples:

Binary Number	Math Equivalent	Decimal Result
10010110	$2^7 + 0 + 0 + 2^4 + 0 + 2^2 + 2^1 + 0$	150
1111	$2^3 + 2^2 + 2^1 + 2^0$	15
101101	$2^5 + 0 + 2^3 + 2^2 + 0 + 2^0$	45

Please find further resources on binary to decimal conversion here:

- Binary to Decimal video
- Converter (to double check your numbers)

Assignment:

You'll be given a starting point of `stack.h` – your job is to complete all of the functions in a `stack.cpp` file of your creation. Do NOT change the header file – your functions must adhere to the standards set in the header. Note: the remove function returns an item. This should be returning the data that was contained at the node that was deleted. Create the add/remove functions in a way that adheres to STACK principles.

`main.cpp`:

This is where you'll need to use your stack to perform binary conversion. A lot of this design wise is left up to you (you can use functions to aid you, or not). However, you must fulfill these items:

- Ask for an initial binary number
- Display the finalized conversion result
- Ask the user if they'd like to enter another binary
    - Validate a yes/no answer
- Loop accordingly

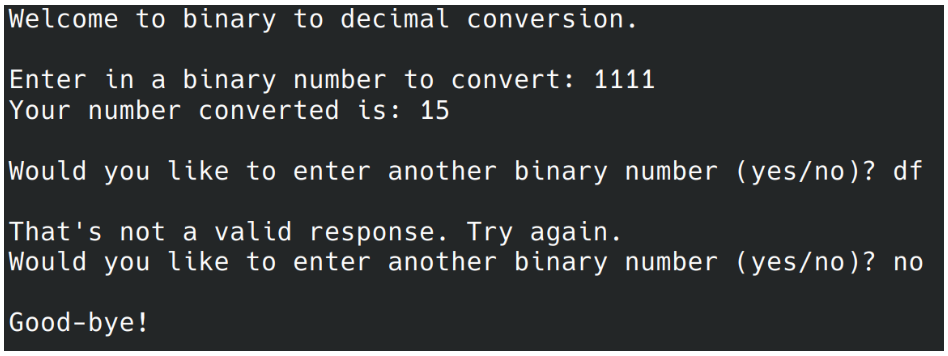
Notice, the conversion itself will happen in main. You're using the stack to aid you in doing so. To be compatible with the Stack class, you should be taking your user input in as a string and iterating through each character to add an individual character to your stack.

Hint:

In order to do exponents, you can use the cmath library through: `#include <cmath>`

Details on the function you'll need to use can be found here.

Example Execution:



```
Welcome to binary to decimal conversion.

Enter in a binary number to convert: 1111
Your number converted is: 15

Would you like to enter another binary number (yes/no)? df

That's not a valid response. Try again.
Would you like to enter another binary number (yes/no)? no

Good-bye!
```

Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                        NetID: <your netID>
 Date: <current date>                     Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Deliverables:

- C++ code
  - Stack class (.h and .cpp files)
  - Main file (.cpp file)
- A document (.pdf) with a screenshot showing the following – if it's too large, feel free to split it into multiple screenshots
  - `00101010` (should equate to 42)
  - `1110` (should equate to 14)
  - `00100101` (should equate to 37)
  - `11111111` (should equate to 255)

Point Breakdown:
(100 points total)
A submission that doesn't contain any code will receive a 0.

- 10pts - stacks implemented correctly
- 30pts - stack.cpp
  - 10pts - constructor/destructor (5pts each)
  - 20pts - add/remove (10pts each)
- 30pts - main.cpp
  - 10pts - correctly uses the stack (adding / removing for converting)
  - 10pts - repetition loop and verification (5pts each)
  - 10pts - conversion is correct
- 20pts - execution screenshots
  - 5pts per binary given (*view deliverables for required text*)
- 10pts - programming style *
- (penalty) -15pts - changing code given to you

* Programming style includes good commenting, variable nomenclature, good whitespace, etc