CompSci 165 Project #2
MAJORITY

due May 9 (W week 6)

Write a routine in C that, using as few 4-element queries as possible,
finds the index of an element that is a majority element in a private array of Boolean-valued "marbles", or indicates that there is no majority element.

Can you use fewer queries than used by other CS 165 teams?

You are required to use the provided `QCOUNT()` subroutine that makes (and counts) element queries.

Requirements

1. Implement a subroutine `int mysub( int n )` to
    - Define a 4-element array, `myarray`
    - Make a sequence of queries, each time invoking `QCOUNT( 1, myarray[] )`
        - `QCOUNT( 1, myarray[] )` queries the **four** `marble[]` elements at indices `myarray[0]`, ..., `myarray[3]`
          (*i.e.*, the contents of `marble[myarray[0]]`, ... , `marble[myarray[3]]`)
          and determines the absolute value of the discrepancy between the number of Boolean 1's and the number of Boolean 0's among those four elements
        - The return value from invoking `QCOUNT` is
            - negative — when any of the four indices stored in `myarray` are out of range or if there are any duplicated indices stored in `myarray`
            - 0 — when the four `marble` values consist of two Boolean 1's and two Boolean 0's
            - 2 — when there are three of one value and one of the other
            - 4 — when all four Boolean values are the same
        - For example,
            - if `marble[]` (indexed from 1 to 7) contains the values: `0,1,1,0,0,0,1` and `myarray`[0] = 3, `myarray`[1] = 2, `myarray`[2] = 1, `myarray`[3] = 7
            - then invoking `QCOUNT( 1, myarray )` should return 2
              because the `marble` cells at indices (3,2,1,7) contain values (1,1,0,1)
              and the two Boolean values (`1` and `0`) occur three times and one time, giving an absolute discrepancy of 2
    - The problem is to determine the location of a majority element,
      and so your routine's return value will be either
      (a) an index in the range 1 to `n` of a majority `marble` element,
      i.e., a marble element whose value is a majority of the `marble[]` element values, or
      (b) 0, when there is no majority, *i.e.*, there are an equal number of marbles having value 0 and having value 1
      (c) -1, if any errors occurred
        - Examples

        - if the `marble` array (indexed from 1 to 7) contains the values: `0,1,1,0,0,0,1`
          then `0` is the majority value (since there are four 0's and three 1's),
          and therefore `mysub()` should return any one of the indices: 1,4,5,6
        - if the `marble` array (indexed from 1 to 6) contains the values: `1,1,0,0,0,1`
          then there is no majority value (since there are three 1's and three 0's),
          and therefore `mysub()` should return 0
    - Your `mysub(n)` routine should
        - be capable of handling values of `n` in the range of 10 to 10000
        - be implemented to minimize the **average-case** number of queries
2. Use the provided main routine which does the following
    - Calls the `mysub( n )` routine 10000 times for each of `n` = 17, 18, 19, 20, 200, and 2000
      and stores the return value in `answer`
    - Invokes `QCOUNT( 0, n )` just prior to each call of `mysub()`
        - This initializes a private array, `marble[]`, that contains `n` random Boolean-valued elements (indexed from 1 to `n`)
        - `n` must be at least 10 and at most 10000
        - `QCOUNT(0,n)` returns 0 normally, but returns -1 if `n` is out of range
    - Invokes `QCOUNT( 2, answer )` after each call of `mysub()`
      This
        - checks whether your routine's answer is correct
        - returns the number of 4-element queries performed if `answer` is correct
        - returns a negative value if `answer` is wrong
    - Outputs the worst case and average (to two decimal places) number of 4-element queries performed for each value of `n`
3. State the worst case (WC) and average number of 4-element queries performed by your `mysub( n )` routine
   as a function of `n`, including all constants (do not use O-notation)
    - first, on the basis of empirical observations (including runs using other values of `n`)
    - second, on the basis of algorithmic theory (WC, expected WC, and average case)
    - justify your statements
    - explain any discrepancies between theoretical predictions and empirical observations

Deliverables

- `mysub.c`
- If you have any, output produced from runs using other values of `n`
- Worst case and average case analysis (statements and justifications)

Evaluation Process

The grader will perform the following tasks

- Place your `mysub.c` file in his directory that contains the class-supplied files: `MAIN.c` and `QCOUNT.c`
- Compile the program on Linux using the command: `gcc MAIN.c -o MAIN`
- View the source code (check for malicious or "cheat" code)

- Run the program using the command: `MAIN > output`
- If the program does not finish execution within three minutes then the grader will abort the execution
- Read analysis document
- Test whether `mysub` works when `n` = 17, 18, 19, 20, 200, 2000
- Compare performance with those of programs from other teams and the Professor
- Points awarding criteria