Microsoft Q# Coding Contest - Winter 2019 - Warmup

G1. AND oracle

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a quantum oracle on N qubits which implements the following function:

$$f(\vec{x}) = x_0 \wedge x_1 \wedge \cdots \wedge x_{N-1}$$

You have to implement an operation which takes the following inputs:

- an array of N ($1 \leq N \leq 8$) qubits x in an arbitrary state (input register),
- a qubit y in an arbitrary state (output qubit),

and performs a transformation $|x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$. The operation doesn't have an output; its "output" is the state in which it leaves the qubits.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (x : Qubit[], y : Qubit) : Unit {
        body (...) {
            // your code here
        }
        adjoint auto;
    }
}
```

G2. OR oracle

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a quantum oracle on N qubits which implements the following function:

$$f(\vec{x}) = x_0 \vee x_1 \vee \cdots \vee x_{N-1}$$

You have to implement an operation which takes the following inputs:

- an array of N ($1 \leq N \leq 8$) qubits x in an arbitrary state (input register),
- a qubit y in an arbitrary state (output qubit),

and performs a transformation $|x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$. The operation doesn't have an output; its "output" is the state in which it leaves the qubits.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (x : Qubit[], y : Qubit) : Unit {
        body (...) {
            // your code here
        }
        adjoint auto;
    }
}
```

G3. Palindrome checker oracle

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a quantum oracle on N qubits which checks whether the vector $\vec{x}$ is a palindrome (i.e., implements the function $f(\vec{x}) = 1$ if $\vec{x}$ is a palindrome, and 0 otherwise).

You have to implement an operation which takes the following inputs:

- an array of N ($1 \leq N \leq 8$) qubits x in an arbitrary state (input register),
- a qubit y in an arbitrary state (output qubit),

and performs a transformation $|x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$. The operation doesn't have an output; its "output" is the state in which it leaves the qubits.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (x : Qubit[], y : Qubit) : Unit {
        body (...) {
            // your code here
        }
        adjoint auto;
    }
}
```

U1. Anti-diagonal unitary

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a unitary operation on N qubits which is represented by an anti-diagonal matrix (a square matrix of size 2^N which has non-zero elements on the diagonal that runs from the top right corner to the bottom left corner and zero elements everywhere else).

For example, for $N = 2$ the matrix of the operation should have the following shape:

```
...X
..X.
.X..
X...
```

Here X denotes a "non-zero" element of the matrix (a complex number which has the square of the absolute value greater than or equal to 10^{-5}), and `.` denotes a "zero" element of the matrix (a complex number which has the square of the absolute value less than 10^{-5}).

The row and column indices of the matrix follow little endian format: the least significant bit of the index is stored first in the qubit array. Thus, the first column of the matrix gives you the coefficients of the basis states you'll get if you apply the unitary to the $|00..0\rangle$ basis state, the second column - to the $|10..0\rangle$ basis state etc. You can use the DumpUnitary tool to get the coefficients of the matrix your unitary implements (up to relative phases between columns) and the corresponding pattern of Xs and `.`s.

You have to implement an operation which takes an array of N ($2 \leq N \leq 5$) qubits as an input and applies the unitary transformation with the matrix of the described shape to it. If there are multiple unitaries which satisfy the requirements, you can implement any of them. The "output" of your operation is the pattern of the matrix coefficients implemented by it; you can see the testing harness in the UnitaryPatterns kata.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[]) : Unit {
        // your code here
    }
}
```

U2. Chessboard unitary

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a unitary operation on N qubits which is represented by a square matrix of size 2^N in which zero and non-zero elements form a chessboard pattern with alternating 2x2 squares (top left square formed by non-zero elements).

For example, for $N = 3$ the matrix of the operation should have the following shape:

```
XX..XX..
XX..XX..
..XX..XX
..XX..XX
XX..XX..
XX..XX..
..XX..XX
..XX..XX
```

Here X denotes a "non-zero" element of the matrix (a complex number which has the square of the absolute value greater than or equal to 10^{-5}), and `.` denotes a "zero" element of the matrix (a complex number which has the square of the absolute value less than 10^{-5}).

The row and column indices of the matrix follow little endian format: the least significant bit of the index is stored first in the qubit array. Thus, the first column of the matrix gives you the coefficients of the basis states you'll get if you apply the unitary to the $|00..0\rangle$ basis state, the second column - to the $|10..0\rangle$ basis state etc. You can use the DumpUnitary tool to get the coefficients of the matrix your unitary implements (up to relative phases between columns) and the corresponding pattern of Xs and `.`s.

You have to implement an operation which takes an array of N ($2 \leq N \leq 5$) qubits as an input and applies the unitary transformation with the matrix of the described shape to it. If there are multiple unitaries which satisfy the requirements, you can implement any of them. The "output" of your operation is the pattern of the matrix coefficients implemented by it; you can see the testing harness in the UnitaryPatterns kata.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[]) : Unit {
        // your code here
    }
}
```

U3. Block unitary

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a unitary operation on N qubits which is represented by the following square matrix of size 2^N:

- top right and bottom left quarters are filled with zero elements,
- top left quarter is an anti-diagonal matrix of size 2^{N-1},
- bottom right quarter is filled with non-zero elements.

For example, for $N = 2$ the matrix of the operation should have the following shape:

```
.X..
X...
..XX
..XX
```

Here X denotes a "non-zero" element of the matrix (a complex number which has the square of the absolute value greater than or equal to 10^{-5}), and `.` denotes a "zero" element of the matrix (a complex number which has the square of the absolute value less than 10^{-5}).

The row and column indices of the matrix follow little endian format: the least significant bit of the index is stored first in the qubit array. Thus, the first column of the matrix gives you the coefficients of the basis states you'll get if you apply the unitary to the $|00..0\rangle$ basis state, the second column - to the $|10..0\rangle$ basis state etc. You can use the DumpUnitary tool to get the coefficients of the matrix your unitary implements (up to relative phases between columns) and the corresponding pattern of Xs and `.`s.

You have to implement an operation which takes an array of N ($2 \leq N \leq 5$) qubits as an input and applies the unitary transformation with the matrix of the described shape to it. If there are multiple unitaries which satisfy the requirements, you can implement any of them. The "output" of your operation is the pattern of the matrix coefficients implemented by it; you can see the testing harness in the UnitaryPatterns kata.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[]) : Unit {
        // your code here
    }
}
```
