Microsoft Q# Coding Contest - Winter 2019

A1. Generate state |00⟩ + |01⟩ + |10⟩

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

You are given two qubits in state $|00\rangle$. Your task is to create the following state on them:

$$\frac{1}{\sqrt{3}}\big(|00\rangle + |01\rangle + |10\rangle\big)$$

You have to implement an operation which takes an array of 2 qubits as an input and has no output. The "output" of your solution is the state in which it left the input qubits.

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
    }
}
```

A2. Generate equal superposition of four basis states

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

You are given N qubits in the zero state $|0...0\rangle$. You are also given four distinct bit vectors which describe four different basis states on N qubits $|\psi_i\rangle$.

Your task is to generate a state which is an equal superposition of the given basis states:

$$|S\rangle = \frac{1}{2}\big(|\psi_0\rangle + |\psi_1\rangle + |\psi_2\rangle + |\psi_3\rangle\big)$$

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

- an array of N ($2 \le N \le 16$) qubits,
- a two-dimensional array of Boolean values $bits$ representing the basis states $|\psi_i\rangle$. $bits$ will have exactly 4 elements, each of $bits[i]$ describing the basis state $|\psi_i\rangle$. Each of $bits[i]$ will have N elements, $bits[i][j]$ giving the state of qubit j in the basis state $|\psi_i\rangle$. Bit values true and false corresponding to $|1\rangle$ and $|0\rangle$ states, respectively; for example, for $N = 2$ an array `[false, true]` will represent the basis state $|01\rangle$. Each pair of $bits[i]$ and $bits[j]$ will differ in at least one position for $i \neq j$.

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 (qs : Qubit[], bits : Bool[][]) : Unit {
        // your code here
    }
}
```

B1. Distinguish three-qubit states

time limit per test: 1 second

memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given 3 qubits which are guaranteed to be in one of the two states:

- $|\psi_0\rangle = \frac{1}{\sqrt{3}}\big(|100\rangle + \omega|010\rangle + \omega^2|001\rangle\big)$, or
- $|\psi_1\rangle = \frac{1}{\sqrt{3}}\big(|100\rangle + \omega^2|010\rangle + \omega|001\rangle\big)$, where $\omega = e^{2i\pi/3}$.

Your task is to perform necessary operations and measurements to figure out which state it was and to return 0 if it was $|\psi_0\rangle$ state or 1 if it was $|\psi_1\rangle$ state. The state of the qubits after the operations does not matter.

You have to implement an operation which takes an array of 3 qubits as an input and returns an integer. Your code should have the following signature:

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

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

B2. Not A, not B or not C?

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

You are given a qubit which is guaranteed to be in one of the following states:

- $|A\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)$,
- $|B\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + \omega|1\rangle\big)$, or
- $|C\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + \omega^2|1\rangle\big)$, where $\omega = e^{2i\pi/3}$.

These states are not orthogonal, and thus can not be distinguished perfectly. Your task is to figure out in which state the qubit is *not*. More formally:

- If the qubit was in state $|A\rangle$, you have to return 1 or 2.
- If the qubit was in state $|B\rangle$, you have to return 0 or 2.
- If the qubit was in state $|C\rangle$, you have to return 0 or 1.
- In other words, return 0 if you're sure the qubit was *not* in state $|A\rangle$, return 1 if you're sure the qubit was *not* in state $|B\rangle$, and return 2 if you're sure the qubit was *not* in state $|C\rangle$.

Your solution will be called 1000 times, each time the state of the qubit will be chosen as $|A\rangle$, $|B\rangle$ or $|C\rangle$ with equal probability. The state of the qubit after the operations does not matter.

You have to implement an operation which takes a qubit as an input and returns an integer. Your code should have the following signature:

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

    operation Solve (q : Qubit) : Int {
        // your code here
    }
}
```

C1. Alternating bits oracle

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Implement a quantum oracle on N qubits which checks whether the bits in the input vector $\vec{x}$ alternate (i.e., implements the function $f(\vec{x}) = 1$ if $\vec{x}$ does not have a pair of adjacent bits in state 00 or 11).

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

- an array of N ($1 \leq N \leq 7$) 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. Note that the input register x has to remain unchanged after applying the operation.

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;
    }
}
```

Note: the operation has to have an adjoint specified for it; `adjoint auto` means that the adjoint will be generated automatically. For details on adjoint, see Operation Definitions.

You are not allowed to use measurements in your operation.

C2. "Is the bit string periodic?" oracle

time limit per test: 4 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

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

A bit string of length N is considered periodic with period P ($1 \leq P \leq N - 1$) if for all $i \in [0, N - P - 1]$ $x_i = x_{i+P}$. Note that P does not have to divide N evenly; for example, bit string `"01010"` is periodic with period $P = 2$.

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

- an array of N ($2 \leq N \leq 7$) 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. Note that the input register x has to remain unchanged after applying the operation.

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;
    }
}
```

Note: the operation has to have an adjoint specified for it; `adjoint auto` means that the adjoint will be generated automatically. For details on adjoint, see Operation Definitions.

You are not allowed to use measurements in your operation.

C3. "Is the number of ones divisible by 3?" oracle

time limit per test: 7 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Implement a quantum oracle on N qubits which checks whether the number of bits equal to 1 in the input vector $\vec{x}$ is divisible by 3

(i.e., implements the function $f(\vec{x}) = 1$ if the number of $x_i = 1$ in $\vec{x}$ is divisible by 3, and 0 otherwise).

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

- an array of N ($1 \leq N \leq 9$) qubits x in an arbitrary state (input register),
- a qubit y in an arbitrary state (output qubit),

and performs a transformation $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$. The operation doesn't have an output; its "output" is the state in which it leaves the qubits. Note that the input register x has to remain unchanged after applying the operation.

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;
    }
}
```

Note: the operation has to have an adjoint specified for it; `adjoint auto` means that the adjoint will be generated automatically. For details on adjoint, see Operation Declarations.

You are not allowed to use measurements in your operation.

D1. Block diagonal matrix

time limit per test: 2 seconds
memory limit per test: 1024 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 which has 2x2 blocks of non-zero elements on the main diagonal and zero elements everywhere else.

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

```
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
    }
}
```

You are not allowed to use measurements in your operation.

D2. Pattern of increasing blocks

time limit per test: 2 seconds
memory limit per test: 1024 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 defined as follows:

- top right and bottom left quarters are filled with zero elements,
- the top left quarter is the same pattern of size 2^{N-1} (for $N = 1$, the top left quarter is a non-zero element),
- the bottom right quarter is filled with non-zero elements.

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

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

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
    }
}
```

You are not allowed to use measurements in your operation.

D3. X-wing fighter

time limit per test: 2 seconds
memory limit per test: 1024 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 which has non-zero elements on both main diagonal and anti-diagonal and zero elements everywhere else.

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

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

You are not allowed to use measurements in your operation.

D4. TIE fighter

time limit per test: 2 seconds
memory limit per test: 1024 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 which has non-zero elements in the following positions:

- the central 2x2 sub-matrix,
- the diagonals of the top right and bottom left square sub-matrices of size $2^{N-1} - 1$ that do not overlap with the central 2x2 sub-matrix,
- the anti-diagonals of the top left and bottom right square sub-matrices of size $2^{N-1} - 1$ that do not overlap with the central 2x2 sub-matrix.

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

```
..X..X..
.X....X.
X......X
...XX...
...XX...
X......X
.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
  }
}
```
You are not allowed to use measurements in your operation.

D5. Creeper

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Implement a unitary operation on 3 qubits which is represented by a square matrix of size 8 with the following pattern:

```
XX....XX
XX....XX
...XX...
...XX...
..X..X..
..X..X..
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 3 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
  }
}
```
You are not allowed to use measurements in your operation.

D6. Hessenberg matrix

time limit per test: 2 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Implement a unitary operation on N qubits which is represented by an upper Hessenberg matrix (a square matrix of size 2^N which has all zero elements below the first subdiagonal and all non-zero elements on the first subdiagonal and above it).

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

```
XXXX
XXXX
.XXX
..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 ($2 \leq N \leq 4$) 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
    }
}
```

You are not allowed to use measurements in your operation.
