Microsoft Q# Coding Contest - Summer 2018

A1. Generate superposition of all 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 ($1 \le N \le 8$) in zero state $|0...0\rangle$.

Your task is to generate an equal superposition of all 2^N basis vectors on N qubits:

$$|S\rangle = \frac{1}{\sqrt{2^N}}\big(|0...0\rangle + ... + |1...1\rangle\big)$$

For example,

- for $N = 1$, the required state is simply $|+\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)$,
- for $N = 2$, the required state is $\frac{1}{2}\big(|00\rangle + |01\rangle + |10\rangle + |11\rangle\big)$.

You have to implement an operation which takes an array of N qubits as an input and has no output. The "output" of the operation 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[]) : ()
    {
        body
        {
            // your code here
        }
    }
}
```

A2. Generate superposition of zero state and a basis state

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

You are given N qubits ($1 \le N \le 8$) in zero state $|0...0\rangle$. You are also given a bitstring $bits$ which describes a non-zero basis state on N qubits $|\psi\rangle$.

Your task is to generate a state which is an equal superposition of $|0...0\rangle$ and the given basis state:

$$|S\rangle = \frac{1}{\sqrt{2}}\big(|0...0\rangle + |\psi\rangle\big)$$

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

- an array of qubits qs,
- an arrays of boolean values $bits$ representing the basis state $|\psi\rangle$. This array will have the same length as the array of qubits. The first element of this array $bits[0]$ will be `true`.

The operation doesn't have an output; its "output" is the state in which it leaves the qubits.

An array of boolean values represents a basis state as follows: the i-th element of the array is true if the i-th qubit is in state $|1\rangle$, and false if it is in state $|0\rangle$. For example, array `[true; false]` describes 2-qubit state $|\psi\rangle = |10\rangle$, and in this case the resulting state should be $\frac{1}{\sqrt{2}}\big(|00\rangle + |10\rangle\big) = |+\rangle \otimes |0\rangle$.

Your code should have the following signature:

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

```
    open Microsoft.Quantum.Canon;

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

A3. Generate superposition of two 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 ($1 \le N \le 8$) in zero state $|0...0\rangle$. You are also given two bitstrings $bits0$ and $bits1$ which describe two different basis states on N qubits $|\psi_0\rangle$ and $|\psi_1\rangle$.

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

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

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

- an array of qubits qs,
- two arrays of Boolean values $bits0$ and $bits1$ representing the basis states $|\psi_0\rangle$ and $|\psi_1\rangle$. These arrays will have the same length as the array of qubits. $bits0$ and $bits1$ will differ in at least one position.

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[], bits0 : Bool[], bits1 : Bool[]) : ()
    {
        body
        {
            // your code here
        }
    }
}
```

A4. Generate W state

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

You are given $N = 2^k$ qubits ($0 \le k \le 4$) in zero state $|0...0\rangle$. Your task is to create a generalized W state on them. Generalized W state is an equal superposition of all basis states on N qubits that have Hamming weight equal to 1:

$$|W_N\rangle = \frac{1}{\sqrt{N}}\big(|100...0\rangle + |010...0\rangle + ... + |00...01\rangle\big)$$

For example, for $N = 1, |W_1\rangle = |1\rangle$.

You have to implement an operation which takes an array of N qubits as an input and has no output. The "output" of the operation 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[]) : ()
    {
        body
        {
            // your code here
        }
    }
}
```

B1. Distinguish zero state and W state

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

You are given N qubits ($2 \leq N \leq 8$) which are guaranteed to be in one of the two states:

- $|0...0\rangle$ state, or
- $|W\rangle = \frac{1}{\sqrt{N}}\big(|100...0\rangle + |010...0\rangle + ... + |00...01\rangle\big)$ state.

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

You have to implement an operation which takes an array of N 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
    {
        body
        {
            // your code here
        }
    }
}
```

B2. Distinguish GHZ state and W state

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

You are given N qubits ($2 \leq N \leq 8$) which are guaranteed to be in one of the two states:

- $|GHZ\rangle = \frac{1}{\sqrt{2}}\big(|0...0\rangle + |1...1\rangle\big)$ state, or
- $|W\rangle = \frac{1}{\sqrt{N}}\big(|100...0\rangle + |010...0\rangle + ... + |00...01\rangle\big)$ state.

Your task is to perform necessary operations and measurements to figure out which state it was and to return 0 if it was GHZ state or 1 if it was W state. The state of the qubits after the operations does not matter.

You have to implement an operation which takes an array of N 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
    {
        body
        {
            // your code here
        }
    }
```

```
    }
```

B3. Distinguish four 2-qubit states

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

You are given 2 qubits which are guaranteed to be in one of the four orthogonal states:

- $|S_0\rangle = \frac{1}{2}\big(|00\rangle + |01\rangle + |10\rangle + |11\rangle\big)$
- $|S_1\rangle = \frac{1}{2}\big(|00\rangle - |01\rangle + |10\rangle - |11\rangle\big)$
- $|S_2\rangle = \frac{1}{2}\big(|00\rangle + |01\rangle - |10\rangle - |11\rangle\big)$
- $|S_3\rangle = \frac{1}{2}\big(|00\rangle - |01\rangle - |10\rangle + |11\rangle\big)$

Your task is to perform necessary operations and measurements to figure out which state it was and to return the index of that state (0 for $|S_0\rangle$, 1 for $|S_1\rangle$ etc.). The state of the qubits after the operations does not matter.

You have to implement an operation which takes an array of 2 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
    {
        body
        {
            // your code here
        }
    }
}
```

B4. Distinguish four 2-qubit states - 2

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

You are given 2 qubits which are guaranteed to be in one of the four orthogonal states:

- $|S_0\rangle = \frac{1}{2}\big(\ \ |00\rangle - |01\rangle - |10\rangle - |11\rangle\big)$
- $|S_1\rangle = \frac{1}{2}\big(-|00\rangle + |01\rangle - |10\rangle - |11\rangle\big)$
- $|S_2\rangle = \frac{1}{2}\big(-|00\rangle - |01\rangle + |10\rangle - |11\rangle\big)$
- $|S_3\rangle = \frac{1}{2}\big(-|00\rangle - |01\rangle - |10\rangle + |11\rangle\big)$

Your task is to perform necessary operations and measurements to figure out which state it was and to return the index of that state (0 for $|S_0\rangle$, 1 for $|S_1\rangle$ etc.). The state of the qubits after the operations does not matter.

You have to implement an operation which takes an array of 2 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
    {
        body
        {
            // your code here
        }
    }
}
```

C1. Distinguish zero state and plus state with minimum error

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

You are given a qubit which is guaranteed to be either in $|0\rangle$ state or in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ state.

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

Note that these states are not orthogonal, and thus can not be distinguished perfectly. In each test your solution will be called 1000 times, and your goal is to get a correct answer at least 80% of the times. In each test $|0\rangle$ and $|+\rangle$ states will be provided with 50% probability.

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
    {
        body
        {
            // your code here
        }
    }
}
```

C2. Distinguish zero state and plus state without errors

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

You are given a qubit which is guaranteed to be either in $|0\rangle$ state or in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ state.

Your task is to perform necessary operations and measurements to figure out which state it was and to return 0 if it was a $|0\rangle$ state, 1 if it was $|+\rangle$ state or -1 if you can not decide, i.e., an "inconclusive" result. The state of the qubit after the operations does not matter.

Note that these states are not orthogonal, and thus can not be distinguished perfectly. In each test your solution will be called 10000 times, and your goals are:

- never give 0 or 1 answer incorrectly (i.e., never return 0 if input state was $|+\rangle$ and never return 1 if input state was $|0\rangle$),
- give -1 answer at most 80% of the times,
- correctly identify $|0\rangle$ state at least 10% of the times,
- correctly identify $|+\rangle$ state at least 10% of the times.

In each test $|0\rangle$ and $|+\rangle$ states will be provided with 50% probability.

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
    {
        body
        {
            // your code here
        }
    }
}
```

D1. Oracle for f(x) = b * x mod 2

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: $f(\vec{x}) = \vec{b} \cdot \vec{x} \mod 2 = \sum_{k=0}^{N-1} b_k x_k \mod 2$, where $\vec{b} \in \{0, 1\}^N$ (a vector of N integers, each of which can be 0 or 1).

For an explanation on how this type of quantum oracles works, see Introduction to quantum oracles.

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

- an array of N qubits x in arbitrary state (input register), $1 \leq N \leq 8$,
- a qubit y in arbitrary state (output qubit),
- an array of N integers b, representing the vector $\vec{b}$. Each element of b will be 0 or 1.

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, b : Int[]) : ()
    {
        body
        {
            // your code here
        }
    }
}
```

D2. Oracle for f(x) = b * x + (1 - b) * (1 - x) mod 2

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:

$$f(\vec{x}) = (\vec{b} \cdot \vec{x} + (\vec{1} - \vec{b}) \cdot (\vec{1} - \vec{x})) \mod 2 = \\ = \sum_{k=0}^{N-1} (b_k x_k + (1 - b_k) \cdot (1 - x_k)) \mod 2$$

Here $\vec{b} \in \{0, 1\}^N$ (a vector of N integers, each of which can be 0 or 1), and $\vec{1}$ is a vector of N 1s.

For an explanation on how this type of quantum oracles works, see Introduction to quantum oracles.

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

- an array of N qubits x in arbitrary state (input register), $1 \leq N \leq 8$,
- a qubit y in arbitrary state (output qubit),
- an array of N integers b, representing the vector $\vec{b}$. Each element of b will be 0 or 1.

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, b : Int[]) : ()
    {
        body
        {
            // your code here
        }
    }
```

```
}
```

D3. Oracle for majority function

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

Implement a quantum oracle on 3 qubits which implements a majority function. Majority function on 3-bit vectors is defined as follows: $f(\vec{x}) = 1$ if vector $\vec{x}$ has two or three 1s, and 0 if it has zero or one 1s.

For an explanation on how this type of quantum oracles works, see Introduction to quantum oracles.

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

- an array of 3 qubits x in arbitrary state (input register),
- a qubit y in arbitrary state (output qubit).

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) : ()
    {
        body
        {
            // your code here
        }
    }
}
```

E1. Bernstein-Vazirani algorithm

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

You are given a quantum oracle - an operation on $N+1$ qubits which implements a function $f : \{0,1\}^N \to \{0,1\}$. You are guaranteed that the function f implemented by the oracle is scalar product function (oracle from problem D1):

$$f(\vec{x}) = \vec{b} \cdot \vec{x} \mod 2 = \sum_{k=0}^{N-1} b_k x_k \mod 2$$

Here $\vec{b} \in \{0,1\}^N$ (an array of N integers, each of which can be 0 or 1).

Your task is to reconstruct the array $\vec{b}$. Your code is allowed to call the given oracle only once.

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

- an integer N - the number of qubits in the oracle input ($1 \le N \le 8$),
- an oracle Uf, implemented as an operation with signature `((Qubit[], Qubit) => ())`, i.e., an operation which takes as input an array of qubits and an output qubit and has no output.

The return of your operation is an array of integers of length N, each of them 0 or 1.

Your code should have the following signature:

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

    operation Solve (N : Int, Uf : ((Qubit[], Qubit) => ())) : Int[]
    {
        body
        {
            // your code here
        }
```

```
    }
}
```

E2. Another array reconstruction algorithm

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

You are given a quantum oracle - an operation on $N+1$ qubits which implements a function $f : \{0,1\}^N \to \{0,1\}$. You are guaranteed that the function f implemented by the oracle can be represented in the following form (oracle from problem D2):

$$f(\vec{x}) = (\vec{b} \cdot \vec{x} + (\vec{1} - \vec{b}) \cdot (\vec{1} - \vec{x})) \mod 2 = \\ = \sum_{k=0}^{N-1} (b_k x_k + (1 - b_k) \cdot (1 - x_k)) \mod 2$$

Here $\vec{b} \in \{0,1\}^N$ (a vector of N integers, each of which can be 0 or 1), and $\vec{1}$ is a vector of N 1s.

Your task is to reconstruct the array $\vec{b}$ which could produce the given oracle. Your code is allowed to call the given oracle only once.

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

- an integer N - the number of qubits in the oracle input ($1 \le N \le 8$),
- an oracle Uf, implemented as an operation with signature `((Qubit[], Qubit) => ())`, i.e., an operation which takes as input an array of qubits and an output qubit and has no output.

The return of your operation is an array of integers of length N, each of them 0 or 1.

Note that in this problem we're comparing the oracle generated by your return to the oracle Uf, instead of comparing your return to the (hidden) value of $\vec{b}$ used to generate Uf. This means that any kind of incorrect return results in "Runtime Error" verdict, as well as actual runtime errors like releasing qubits in non-zero state.

Your code should have the following signature:

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

    operation Solve (N : Int, Uf : ((Qubit[], Qubit) => ())) : Int[]
    {
        body
        {
            // your code here
        }
    }
}
```