// Part I. Oracles

// 1. f(x) = 0

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

operation Oracle\_Zero (x : Qubit[], y : Qubit) : Unit {

// Task 1.2. f(x) = 1

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

operation Oracle\_One (x : Qubit[], y : Qubit) : Unit

// Task 1.3. f(x) = xₖ (the value of k-th qubit)

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// 3) 0-based index of the qubit from input register (0 <= k < N)

// Goal: transform state |x, y⟩ into state |x, y ⊕ xₖ⟩ (⊕ is addition modulo 2).

operation Oracle\_Kth\_Qubit (x : Qubit[], y : Qubit, k : Int) : Unit {

// Task 1.4. f(x) = 1 if x has odd number of 1s, and 0 otherwise

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

operation Oracle\_OddNumberOfOnes (x : Qubit[], y : Qubit) : Unit {

// Task 1.5. f(x) = Σᵢ rᵢ xᵢ modulo 2 for a given bit vector r (scalar product function)

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// 3) a bit vector of length N represented as Int[]

// You are guaranteed that the qubit array and the bit vector have the same length.

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

//

operation Oracle\_ProductFunction (x : Qubit[], y : Qubit, r : Int[]) : Unit {

// Task 1.6. f(x) = Σᵢ (rᵢ xᵢ + (1 - rᵢ)(1 - xᵢ)) modulo 2 for a given bit vector r

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// 3) a bit vector of length N represented as Int[]

// You are guaranteed that the qubit array and the bit vector have the same length.

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

operation Oracle\_ProductWithNegationFunction (x : Qubit[], y : Qubit, r : Int[]) : Unit {

// Task 1.7. f(x) = Σᵢ xᵢ + (1 if prefix of x is equal to the given bit vector, and 0 otherwise) modulo 2

// Inputs:

// 1) N qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// 3) a bit vector of length P represented as Int[] (1 <= P <= N)

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

//

// A prefix of length k of a state |x⟩ = |x₁, ..., xₙ⟩ is the state of its first k qubits |x₁, ..., xₖ⟩.

// For example, a prefix of length 2 of a state |0110⟩ is 01.

operation Oracle\_HammingWithPrefix (x : Qubit[], y : Qubit, prefix : Int[]) : Unit

// Task 1.8\*. f(x) = 1 if x has two or three bits (out of three) set to 1, and 0 otherwise (majority function)

// Inputs:

// 1) 3 qubits in arbitrary state |x⟩ (input register)

// 2) a qubit in arbitrary state |y⟩ (output qubit)

// Goal: transform state |x, y⟩ into state |x, y ⊕ f(x)⟩ (⊕ is addition modulo 2).

operation Oracle\_MajorityFunction (x : Qubit[], y : Qubit) : Unit