Homework 6
CSE 490q

Due: Mon, Nov 16th by 11pm PST

Instructions:

For this homework clone this GitHub repository on your machine:
https://github.com/anpaz/uw-490q-au20.git
Optionally, you can also download a zip file with the contents using this link:
https://github.com/anpaz/uw-490q-au20/archive/master.zip
In this repository you will find an `hw6` folder, and in it a folder for each one of the problems, named p1, p2 and p3 accordingly.
For each problem there are two Q# files:

- `Program.qs`: with the Q# EntryPoint calling the operation that you need to implement
- `Answers-n.qs`: the file that with the operations you need to implement

For your answers, you only need to modify the later. When submitting your answers to greadescope, please include only this file plus an screenshot that shows your program running.

Please post any questions on the course's message board.

Problem 1

For this problem, please write all your code in:

```
/hw6/p1/Answer-1.qs
```

To run your program you can use:

```
cd /hw6/p1
dotnet run -x 3 -y 2
```

An in-place adder is a quantum operation that given two qubit registers in state $|a\rangle$ and $|b\rangle$, sets the qubits to state $|a\rangle|b \oplus a\rangle$ with an optional carry qubit that indicates overflow.

a. Implement a reversible in-place adder with carry. The adder you implement must be capable of adding 4 qubit registers.
   a. The description of a circuit implementing this operation can be found in section 2 of the "Addition on a Quantum Computer" paper by Drapper (arXiv:quant-ph/0008033)
   b. A great resource to learn how to implement this step-by-step are the first two sections of the RippleCarryAdder Kata from the QuantumKatas repository. If you plan to use this, make sure you read the instructions on how to run the Katas locally from the repos's README file.

b. Implement an in-place subtractor. An in-place subtractor is a quantum operation that given two qubit registers in state $|a\rangle$ and $|b\rangle$, sets the qubits to state $|a\rangle|b \ominus a\rangle$ with a carry qubit that indicates overflow.

Problem 2

For this problem, please write all your code in:

```
/hw6/p2/Answer-2.qs
```

To run your program you can use:

```
cd /hw6/p2
dotnet run -x 6
```

An oracle is a quantum operation that transforms a state $|x\rangle|y\rangle$ into state $|x\rangle|y \oplus f(x)\rangle$. Implement the following oracles:

a. The **Oracle_And:** where $f(x) = x_0 \wedge x_1 \wedge \ldots \wedge x_n$

   Inputs:
   1. n qubits in an arbitrary state $|x\rangle$ (input/query register).
   2. A qubit in an arbitrary state $|y\rangle$ (target qubit).

   Goal:
   Transform state $|x,y\rangle$ into state $|x,y \oplus f(x)\rangle$ ($\oplus$ is addition modulo 2), i.e., flip the target state if all qubits of the query register are in the $|1\rangle$ state, and leave it unchanged otherwise.

   Leave the query register in the same state it started in.

   Signature:
   ```
   operation Oracle_And (queryRegister : Qubit[], target : Qubit) : Unit
   is Adj
   ```

   Hint:
   The unitary corresponding to this oracle is:

   $$U_{and}\, |x\rangle|y\rangle \;=\; \begin{cases} |x\rangle|y\rangle & if\ x \neq 1\ldots1 \\ |x\rangle X|y\rangle & if\ x = 1\ldots1 \end{cases}$$

b. The **Oracle_6:** where $f(x) = 1\ iff\ x\ == \ 6$

   Inputs:
   3. n qubits in an arbitrary state $|x\rangle$ (input/query register).
   4. A qubit in an arbitrary state $|y\rangle$ (target qubit).

   Goal:
   Transform state $|x,y\rangle$ into state $|x,y \oplus f(x)\rangle$ ($\oplus$ is addition modulo 2), i.e., flip the target state if the qubits in the query register are in the $|6\rangle$ state, and leave it unchanged otherwise.

   Leave the query register in the same state it started in.

   Signature:
   ```
   operation Oracle_6 (queryRegister : Qubit[], target : Qubit) : Unit
   is Adj
   ```

   Hint:

The unitary corresponding to this oracle is:

$$U_{and}\,|x\rangle|y\rangle = \begin{cases} |x\rangle|y\rangle & if\ x \neq 0110 \\ |x\rangle X|y\rangle & if\ x = 0110 \end{cases}$$

c. The **`Oracle_Or`:** where $f(x) = x_0 \vee x_1 \vee \quad ... \vee x_n$

Inputs:

1. n qubits in an arbitrary state $|x\rangle$ (input/query register).
2. A qubit in an arbitrary state $|y\rangle$ (target qubit).

Goal:
Transform state $|x,y\rangle$ into state $|x,y\oplus f(x)\rangle$ ($\oplus$ is addition modulo 2), i.e., flip the target state if at least one qubit of the query register is in the $|1\rangle$ state, and leave it unchanged otherwise.

Leave the query register in the same state it started in.

Signature:
```
operation Oracle_Or (queryRegister : Qubit[], target : Qubit) : Unit
is Adj
```

Hint:

The unitary corresponding to this oracle is:

$$U_{and}\,|x\rangle|y\rangle = \begin{cases} |x\rangle|y\rangle & if\ x \neq 0...0 \\ |x\rangle X|y\rangle & if\ x = 0...0 \end{cases}$$

d. The **`Oracle_SATClause`**: where $f(x) = SAT\ clause$

Inputs:

1. N qubits in an arbitrary state $|x\rangle$|x⟩ (input/query register).
2. A qubit in an arbitrary state $|y\rangle$|y⟩ (target qubit).
3. A 1-dimensional array of tuples `clause` which describes one clause of a SAT problem instance `clause(x)`.

`clause` is an array of one or more tuples, each of them describing one component of a SAT clause.

Each tuple is an (Int, Bool) pair:

a. the first element is the index j of the variable x_j,
b. the second element is true if the variable is included as itself (x_j) and false if it is included as a negation ($\neg x_j$).

Example:

- The clause $x_0 \vee \neg x_1$ can be represented as [(0, true), (1, false)].

Goal:
Transform state $|x,y\rangle$ into state $|x,y\oplus f(x)\rangle$ ($\oplus$ is addition modulo 2) if the state of the qubits match the state represented by the SAT clause.

Leave the query register in the same state it started in.

Signature:
```
operation Oracle_SATClause (queryRegister : Qubit[], target : Qubit, clause : (Int,
Bool)[]) : Unit is Adj
```

e. The **`Oracle_SAT`** where $f(x) = SAT\ problem$

Inputs:

1. N qubits in an arbitrary state $|x\rangle$ (input/query register).
2. A qubit in an arbitrary state $|y\rangle$ (target qubit).
3. A 2-dimensional array of tuples `problem` which describes the k-SAT problem instance $f(x)$.
   i-th element of `problem` describes the i-th clause of $f(x)$; it is an array of one or more tuples, each of them describing one component of the clause as described in the previous question.

Example:
A more general case of the OR oracle for 3 variables $f(x)$=($x_0 \vee x_1 \vee x_2$) can be represented as `[[(0, true), (1, true), (2, true)]]`.

Goal:
Transform state $|x,y\rangle$ into state $|x,y \oplus f(x)\rangle$ ($\oplus$ is addition modulo 2).

Leave the query register in the same state it started in.

Signature:
```
operation Oracle_SAT (queryRegister : Qubit[], target : Qubit, problem : (Int, Bool)[][])
: Unit is Adj
```

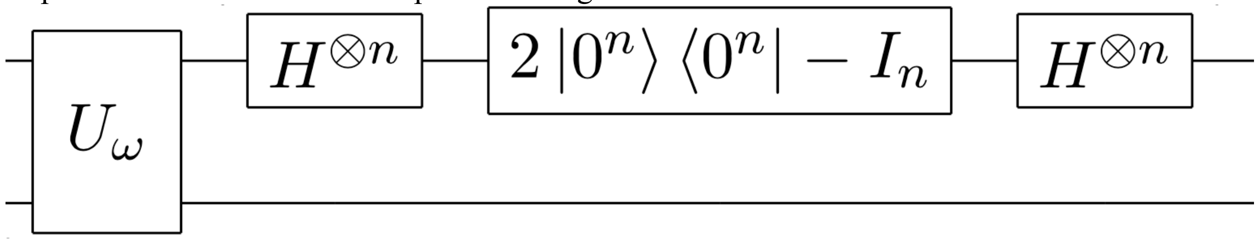
Problem 3

For this problem, please write all your code in:
```
/hw6/p3/Answer-3.qs
```

To run your program you can use:
```
cd /hw6/p3
dotnet run -x 6
```

Grover's algorithm is a quantum algorithm that finds with high probability the unique input to an oracle function that produces a particular output value, using just $O(\sqrt{N})$ evaluations of the function, where N is the size of the function's domain.

a. Implement the Grover Iteration part of the algorithm:



Namely:

a. Apply the Oracle

b. Apply the Hadamard transform
c. Perform a conditional phase shift
d. Apply the Hadamard transform again

Signature:

```
operation GroverIteration (register : Qubit[], oracle : (Qubit[] => Unit is Adj)) : Unit
is Adj { }
```

This iteration will be called as part of the `GroverSearch` operation using the oracles implemented in Problem 2. Take a look in `Program.cs` to understand how `GroverSearch` works and was implemented in Q# calling `GroverIteration`.