A Beginner's Guide to Quantum Computing and Q#

Rate this article ★★★★★

Lee Stott (https://social.msdn.microsoft.com/profile/Lee+Stott) February 6, 2018

1 (https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/a-beginners-guide-to-quantum-computing-and-q/#comments)

Share (https://www.facebook.com/sharer/sharer.php?u=https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/a-beginners-guide-to-quantum-computing-and-q/&src=sdkpreparse) 0

Follow Us

(http://www.twitter.com/lee_stott)

(https://blogs.msdn.microsoft.com/uk_faculty_connection/feed/)

(https://www.linkedin.com/groups/3829706)

(http://imagine.microsoft.com)

(http://aka.ms/azure4students)

(https://imagine.microsoft.com/en-us/productx)

(http://mva.microsoft.com/imagine)

Archives

blog post, Quantum Computing.

So maybe you heard the announcement of the Quantum Development Kit (http://www.microsoft.com/quantum) and thought it sounded insanely cool… Then you had a look and realized you know next to nothing about quantum mechanics. Yep, me too.

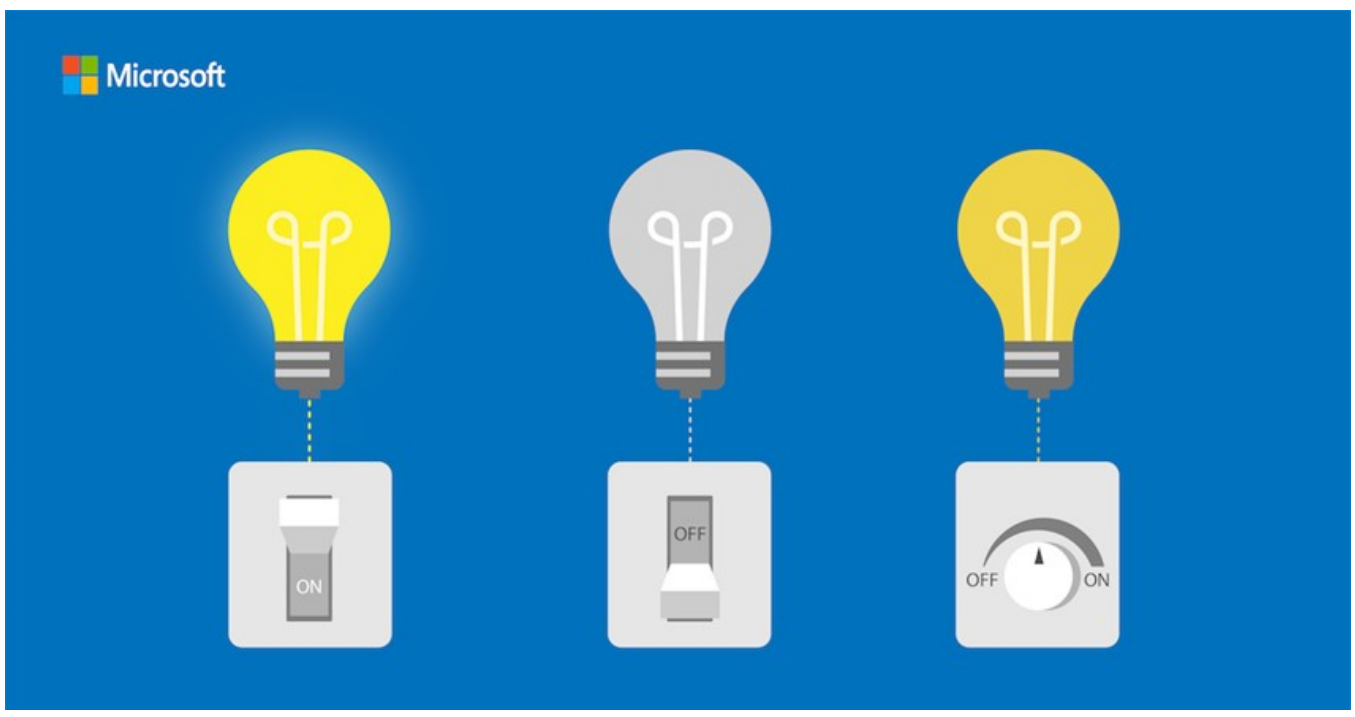
But that's okay. In about 30 minutes you're going to know enough about qubits, superposition and entanglement to write your very first quantum program, and most importantly, have a reasonable idea of what it means. Sounds smashing.

Contents of this blog

Back to Basics

Unless you're a super low-level programmer, it's easy to forget that the programs we write are essentially just manipulating a bunch of 0s and 1s stored in our 'classical' computer. These are discrete, binary states. Quantum computers however operate on *continuous* states – that's part of what makes them so powerful.

So instead of a classical bit being 'on' or 'off' like a light switch, in a quantum computer we have qubits, which are more like a dimmer switch, being any possible combination of 'on' and 'off' in between.



(https://msdnshared.blob.core.windows.net/media/2018/02/image238.png)

We can notate the two qubit states using *Dirac notation* which equates to these vectors:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad \text{and} \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image93.png)

If we add the two states together, we can express any possible combination of |0⟩ and |1⟩, and in quantum mechanics this is called a superposition. The notation for that looks like this:

$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$

Here α and β are kind of like probabilities with the minor difference that they are complex numbers. It doesn't matter if you think of them as real numbers; however if you do, then remember that they'll sometimes be negative, and that the sum of their squares is always 1.

Measuring a Qubit

Now quantum states are weird. If you "look" at a qubit, it immediately collapses its state… a bit like looking at your bed after a long day. What I mean by that is, if we have a qubit that is in a superposition and we measure it, it's either going to collapse to the |0⟩ state or the |1⟩ state (we can't measure both at the same time!). After measurement, we effectively lose the prior values of α and β .

Because of that, when we measure a qubit we talk about the result we get in terms of probabilities. In general, a qubit when measured will give '|0⟩' with probability $|\alpha|^2$, and '|1⟩' (https://msdnshared.blob.core.windows.net/media/2018/02/image149.png) , and '|1⟩' with probability $|\beta|^2$ (https://msdnshared.blob.core.windows.net/media/2018/02/image169.png) . Let's have a look at an example - say we had the following qubit:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

(https://msdnshared.blob.core.windows.net/media/2018/02

$$(\text{i.e. } \alpha = \frac{1}{\sqrt{2}} \text{ and } \beta = \frac{1}{\sqrt{2}})$$

/image94.png)

If we measure the qubit we are likely to get the outcome '0' fifty percent of the time, because $\left|\frac{1}{\sqrt{2}}\right|^2 = 0.5$

(https://msdnshared.blob.core.windows.net/media/2018/02/image95.png), and its post-measurement state will be |0⟩ (i.e. α =1 and β=0). Similarly, we are likely to get the outcome '1' fifty percent of the time, for the same reason, and its post-measurement state will be |1⟩ (i.e. α =0 and β=1).

(https://msdnshared.blob.core.windows.net/media/2018/02/image96.png)

This can seem confusing at first (and the second, third, and fourth time!). The main point here is that these probabilistic quantum states can be used for computation and in some cases, this is much more efficient than classical systems due to their 'quantum weirdness'. Next, we're going to look at how we can manipulate these qubits for computation just like you can with classical bits.

Quantum Gates

We're back in familiar territory. In classical computing we use logic gates to operate on bits, and likewise we can use quantum gates to operate on qubits. Think of the NOT gate, which takes 0 → 1 and 1 → 0. A quantum NOT gate bears some resemblance to its classical brother, as it takes |0⟩ → |1⟩ and |1⟩ → |0⟩. So a qubit in the state α |0⟩ + β |1⟩ becomes α |1⟩ + β |0⟩ after such a gate has operated on it. The not gate can be written as a matrix, *X*, which swaps the roles of 0 and 1 in the state

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image97.png)

As we can see, X|0⟩ = |1⟩ and X|1⟩ = |0⟩:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image98.png)

Because |0⟩ and |1⟩ are defined in vector form as $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ (https://msdnshared.blob.core.windows.net/media/2018/02

/image99.png)and $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image100.png) , you can think of the first column of X as the transformation applied to |0⟩ and the second column as the transformation applied to |1⟩.

Now this doesn't seem all that different to what we're used to. But I want to remind you of what we found in the previous section, that the measurement of a qubit is probabilistic. As we know from basic statistics, all probabilities sum to one. As a result, we have the condition that $|\alpha|^2 + |\beta|^2 = 1$ (https://msdnshared.blob.core.windows.net/media /2018/02/image150.png) for a quantum state α |0⟩ + β |1⟩.

A consequence of this is that there are some constraints on what gates we can have in the quantum world. And one of them is that this normalization condition on the quantum states, $|\alpha|^2 + |\beta|^2 = 1$

(https://msdnshared.blob.core.windows.net/media/2018/02/image151.png) , should hold both before and after the gate has acted. In terms of matrices, this condition will hold if a matrix is *unitary*.

I'm about to tell you what unitary means in maths speak, if we read it *really quickly* we can get to the next sentence. A gate is unitary if , where $U^\dagger U = I$, where $U^\dagger$ i (https://msdnshared.blob.core.windows.net/media/2018/02 /image152.png)s obtained by transposing and then complex conjugating U, and I i

(https://msdnshared.blob.core.windows.net/media/2018/02/image153.png) is the two by two identity matrix. In plain English, it means that the transformation doesn't change the length of the vector. Keeping the length constant is the same as ensuring that all the probabilities add up to 100%. Having probabilities add up to 200% or 25% would make no sense and using unitary matrices guarantees at least this type of craziness is off the menu in the quantum world.

But we can breathe a sigh of relief – this is our only constraint. Some classical gates don't have quantum equivalents for this reason, but there are some new ones that crop up too, and we're going to look at some important ones next.

Important Gates – Bill, Z and Hadamard

The following gates you'll see in our first quantum program, so make a mental note. The *Z* gate is nice and simple, it leaves |0⟩ unaltered and makes |1⟩ negative. That can be written as a matrix like this,

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image101.png)

which transforms the qubit states from |0⟩→|0⟩ and from |1⟩→ -|1⟩. (Remember, the first column corresponds to the transformation applied to |0⟩, and the second column to |1⟩.)

Then we also have the *Hadamard* gate which creates a superposition between the |0⟩ and |1⟩ states, similar to those seen previously. And that can be written as a matrix like this,

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image102.png)

which transforms the qubit states from $|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image103.png)and $|1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$ (https://msdnshared.blob.core.windows.net/media/2018/02/image104.png) .

If you're interested in learning more about unitary matrices, and how to visualize these gates have a look at the Further Resources section where I've listed some useful material.

Multiple Qubits

Time for some more familiar stuff. In the classical world we can have multiple bits, such as 00, 01, 10 and 11. In quantum computing we can similarly have |00⟩, |01⟩, |10⟩ and |11⟩. We can describe the two qubits using a vector,

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle.$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image105.png)

Just like before, the measurement result of '00' for example, occurs with probability $|\alpha_{00}|^2$ (https://msdnshared.blob.core.windows.net/media/2018/02/image106.png),

'01' with probability $|\alpha_{01}|^2$ (https://msdnshared.blob.core.windows.net/media/2018/02/image107.png), and so on.

Now say we wanted to measure the first qubit (instead of both), the measurement result of 0 occurs with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$. (https://msdnshared.blob.core.windows.net/media/2018/02/image108.png) . And remember measurement changes the state, so afterwards the vector would now have a value of

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image109.png)

You can see on the top line we've got rid of all the terms whose first bit was 1, since they weren't consistent with the measurement being 0. The square root in the denominator renormalizes the vector so that the sum of the squared amplitudes is still 1, which we require for it to be a valid quantum state.

Another Important Gate

You've seen the NOT gate, next up we've got the CNOT gate, which stands for *controlled*-NOT. It takes two input qubits, the first is the *control* qubit, and the second is the *target* qubit. As you might have guessed, if the control is |0⟩, then the target qubit is unchanged. If the control is |1⟩, then a NOT is performed on the target qubit.

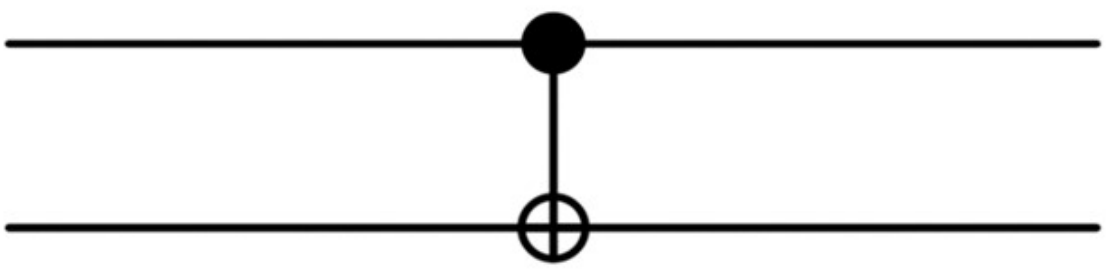
There are a few ways to think of CNOT. Like *X*, *Z* and *H,* we can show it in matrix form, *U*.

$$U_{CN} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$ (https://msdnshared.blob.core.windows.net

/media/2018/02/image110.png)

Looking at the columns of the matrix we can see that they correspond to the following transformations, |00⟩→|00⟩, |01⟩→|01⟩, |10⟩→|11⟩ and |11⟩→|10⟩. As with the other matrices we've looked at U_{CN} (https://msdnshared.blob.core.windows.net/media/2018/02/image111.png), is unitary, meaning $U^{\dagger}{}_{CN} U_{CN} = I$ (https://msdnshared.blob.core.windows.net/media/2018/02/image112.png) .

It can also be drawn like this, where the control bit is on top and the target bit is at the bottom:



(https://msdnshared.blob.core.windows.net/media/2018/02/image113.png)

Looks like something you might find in the Tate Modern.

Bell States

These get a whole section of their own. The Bell states (there are four of them). You'll see one of them ($|\phi^+\rangle$) appear in the quantum program a little later, so here it is:

$$|\phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02

/image114.png)

Suppose we were to measure the first qubit, we're going to get |0⟩ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = 0.5,$

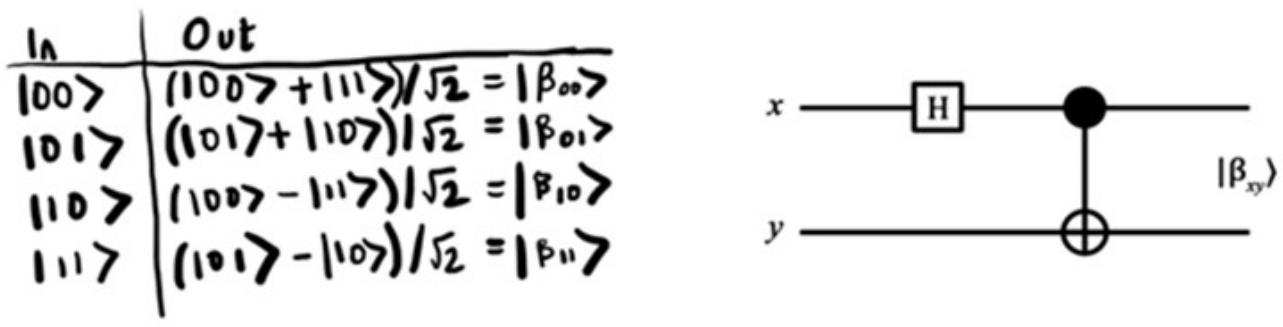
(https://msdnshared.blob.core.windows.net/media/2018/02/image115.png) leaving the post-measurement state $|\psi'\rangle = |00\rangle$, or |1⟩ with the same probability, 0.5, and a post-measurement state $|\psi'\rangle = |11\rangle$. In case you were interested, here's the complete set of four Bell States (which represent the simplest examples of quantum entanglement):

$$|\psi^{\pm}\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}} \qquad |\phi^{\pm}\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}}$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image116.png)

Suppose instead that we had measured the *second* qubit - using the same logic it would leave the post-measurement state |00⟩ or |11⟩. If we then decided to measure the *first* qubit, the probability is no longer a half, we're going to get |0⟩ with probability 1 or 0 depending on the post-measurement state. The key point here is that these outcomes are correlated. This was first noticed by Einstein, Podolsky and Rosen (which is why you might see them called EPR pairs), with further progress made by John Bell.

One final thing, we can generate Bell states using a combination of a Hadamard gate followed by a CNOT. I think this is pretty cool. The Hadamard transforms the first qubit by putting it into a superposition, this is then used by the CNOT as a control to modify the target qubit. That is neatly described by the following circuit diagram:



(https://msdnshared.blob.core.windows.net/media/2018/02/image117.png)

If you want to see the working behind each of those transformations, take a look at the Appendix. For now, with our knowledge of qubit states, quantum gates, and Bell states, we're ready to tackle our first quantum program.

Writing a Quantum Program

For this we're going to follow along with the documentation here:

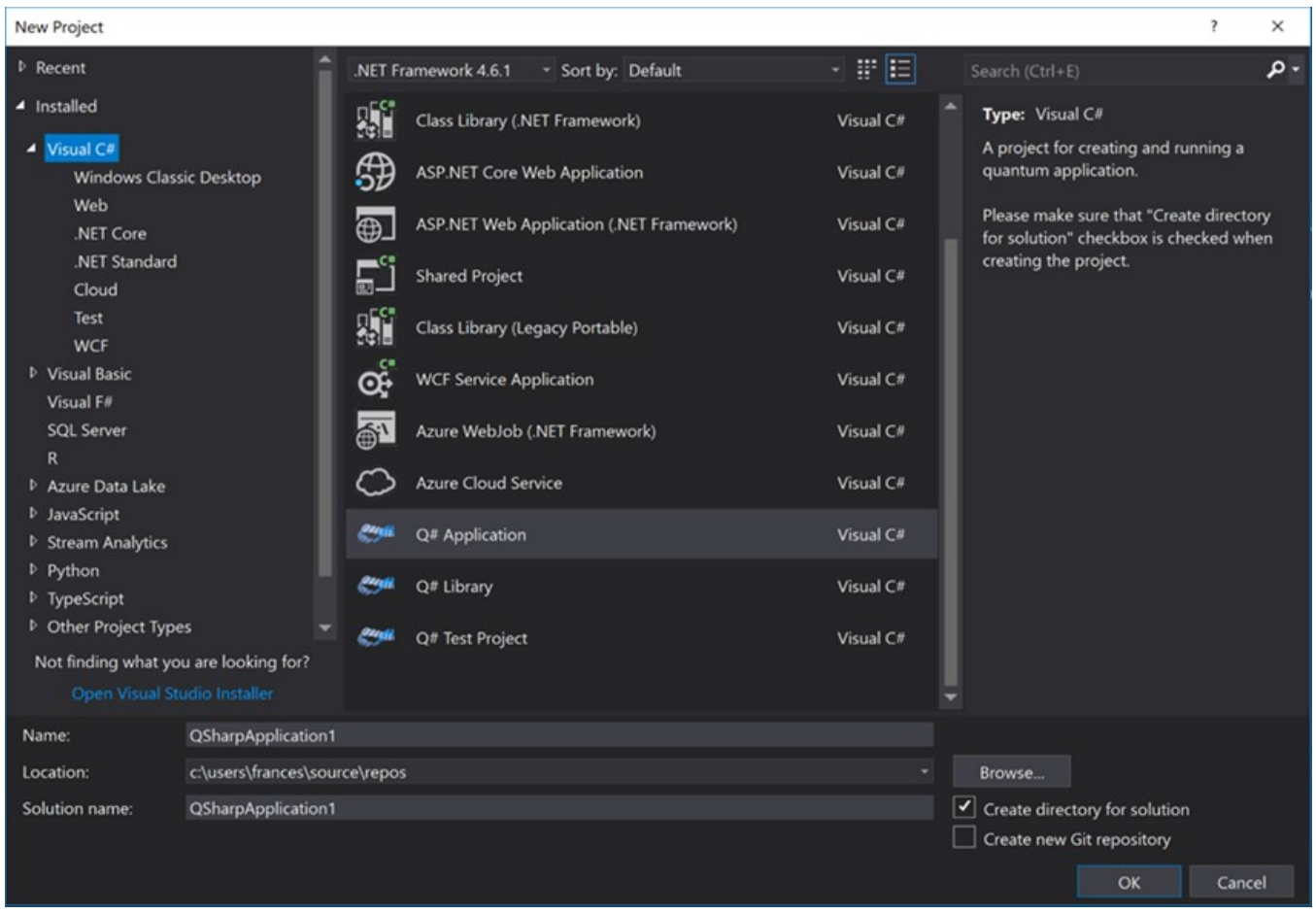
https://docs.microsoft.com/en-us/quantum/quantum-writeaquantumprogram?view=qsharp-preview (https://docs.microsoft.com/en-us/quantum/quantum-writeaquantumprogram?view=qsharp-preview)

The purpose of this tutorial is to get set up with the quantum development kit (steps 1-2), allocate a qubit and do some simple things like setting its value and measuring it (steps 3-5), then we'll superpose a qubit (step 6), before entangling two qubits to create a Bell state or EPR pair (step 7).

I suggest you follow the tutorial linked above, and refer back to the steps listed here for tips and further clarification if you need it.

Step 1: Create a Project and Solution

Q# is hiding at the bottom of the list there.



(https://msdnshared.blob.core.windows.net/media/2018/02/image154.png)

Step 2 (optional): Update NuGet Packages

I did this but you may like to live dangerously.

Step 3: Enter the Q# Code

```
Q#
namespace Quantum.Bell
{
    open Microsoft.Quantum.Primitive;

    operation Set (desired: Result, q1: Qubit) : ()
    {
        body
        {
            let current = M(q1);
            if (desired != current)
            {
                X(q1);
            }
        }
    }
}
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image155.png)

This operation sets our qubit to a desired state of Zero or One (that's for us to choose). First we measure (M) the qubit, which collapses its state to either 0 or 1. If our measured state doesn't match the provided desired state, we use the NOT gate, *X*, to flip it, otherwise we do nothing.

```
operation BellTest (count : Int, initial: Result) : (Int,Int)
{
    body
    {
        mutable numOnes = 0;
        using (qubits = Qubit[1])
        {
            for (test in 1..count)
            {
                Set (initial, qubits[0]);

                let res = M (qubits[0]);

                // Count the number of ones we saw:
                if (res == One)
                {
                    set numOnes = numOnes + 1;
                }
            }
            Set(Zero, qubits[0]);
        }
        // Return number of times we saw a |0> and number of times we saw a |1>
        return (count-numOnes, numOnes);
    }
}
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image156.png)

This little bit of code is going to test the previous operation we just wrote. It really isn't a difficult program, it's just verifying that the qubit was set to the value we wanted.

To do that, it's going to loop count number of times and tot up how many One states we observed, and it stores that in the numOnes variable.

The Qubit[1] syntax simply allocates one qubit to the qubits array, which we index at 0. If we wanted to allocate two qubits (which we'll do later), we'd use Qubit[2] and index qubits at 0 and 1.

In the for loop we set the qubit we've allocated to some initial value One or Zero (we do this explicitly in the **Driver.cs** file coming up). We measure that, and then count if the result was One. We then return how many One and Zero states we saw. We also set the qubit to Zero at the end, so it is left in a known state.

Step 4: Enter the C# Driver Code

```
using (var sim = new QuantumSimulator())
{
    // Try initial values
    Result[] initials = new Result[] { Result.Zero, Result.One };
    foreach (Result initial in initials)
    {
        var res = BellTest.Run(sim, 1000, initial).Result;
        var (numZeros, numOnes) = res;
        System.Console.WriteLine(
            $"Init:{initial,-4} 0s={numZeros,-4} 1s={numOnes,-4}");
    }
}
System.Console.WriteLine("Press any key to continue...");
System.Console.ReadKey();
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image157.png)

In this driver code we construct the quantum simulator, we create an array of initial values we want to test, being Zero and One. We run the simulation with a count of 1000, and then print the results for debugging using System.Console.WriteLine.

Step 5: Build and Run

```
Init:Zero 0s=1000 1s=0
Init:One  0s=0    1s=1000
Press any key to continue...
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image158.png)

All being well, you should have this output. This means that when we set the initial qubit to Zero and repeated this a thousand times we correctly counted 1000 |0⟩s, and the same when we set the initial qubit to One.

Step 6: Creating Superposition

Now we're working up to something a bit more exciting. Here we flip the qubit using a NOT gate.

```
X(qubits[0]);
let res = M (qubits[0]);
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image159.png)

Then we run the program again, and we see that the results are reversed.

```
Init:Zero 0s=0    1s=1000
Init:One  0s=1000 1s=0
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image160.png)

We then replace the NOT gate with a Hadamard gate (H). As we know, this places the qubit in a superposition, where it is in a probabilistic state of both |0⟩ and |1⟩ simultaneously.

```
H(qubits[0]);
let res = M (qubits[0]);
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image161.png)

Running the program again we get some interesting results.

```
Output
Init:Zero 0s=484  1s=516
Init:One  0s=522  1s=478
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image162.png)

We get |0⟩ roughly half the time and |1⟩ half the time.

Step 7: Creating Entanglement

We're going to make a Bell state now. Looking at the code below, we begin by allocating ourselves two qubits using the Qubit[2] syntax. The first qubit (*x* in our earlier circuit diagram) we set to some initial value, and the second qubit (*y* in our diagram) we set to Zero. This is akin to having an input of either |00⟩ or |10⟩, depending on x

(https://msdnshared.blob.core.windows.net/media/2018/02/image147.png)

```
Q#
operation BellTest (count : Int, initial: Result) : (Int,Int)
{
    body
    {
        mutable numOnes = 0;
        using (qubits = Qubit[2])
        {
            for (test in 1..count)
            {
                Set (initial, qubits[0]);
                Set (Zero, qubits[1]);

                H(qubits[0]);
                CNOT(qubits[0],qubits[1]);
                let res = M (qubits[0]);

                // Count the number of ones we saw:
                if (res == One)
                {
                    set numOnes = numOnes + 1;
                }
            }
            Set(Zero, qubits[0]);
            Set(Zero, qubits[1]);
        }
        // Return number of times we saw a |0> and number of times we saw a |1>
        return (count-numOnes, numOnes);
    }
}
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image163.png)

Following the circuit diagram, we apply the Hadamard gate to the first qubit, qubits[0], which puts it in a superposition. We then apply the CNOT gate (using qubits[0]as the control qubit and qubits[1] as the target) and measure the result.

Before we go into the results we expect, let's just have a quick recap on the behaviour of our Bell state. When we measure the first qubit, we're going to get |0⟩ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = 0.5$,

(https://msdnshared.blob.core.windows.net/media/2018/02/image129.png) , leaving the post-measurement state $|\psi'\rangle=|00\rangle$, or |1⟩ with the same probability (0.5), and a post-measurement state $|\psi'\rangle=|11\rangle$. So then when we measure the second qubit, it's going to be |0⟩ if the first qubit was |0⟩, or |1⟩ if the first qubit was |1⟩. In our results we're looking for the second qubit's measurement to 'agree' with the first if we've successfully entangled the two qubits.

You can see that little piece of code in the if statement that we've added below, where we check if the measurement of qubits[1] is equal to the result of measuring qubits[0].

```
Q#
operation BellTest (count : Int, initial: Result) : (Int,Int,Int)
{
    body
    {
        mutable numOnes = 0;
        mutable agree = 0;
        using (qubits = Qubit[2])
        {
            for (test in 1..count)
            {
                Set (initial, qubits[0]);
                Set (Zero, qubits[1]);

                H(qubits[0]);
                CNOT(qubits[0],qubits[1]);
                let res = M (qubits[0]);

                if (M (qubits[1]) == res)
                {
                    set agree = agree + 1;
                }

                // Count the number of ones we saw:
                if (res == One)
                {
                    set numOnes = numOnes + 1;
                }
            }
        Set(Zero, qubits[0]);
        Set(Zero, qubits[1]);
        }
        // Return number of times we saw a |0> and number of times we saw a |1>
        return (count-numOnes, numOnes, agree);
    }
}
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image164.png)

Before we can check the results we need to make a final modification to the **Driver.cs** file, which is to add our agree variable.

C# Copy

```
using (var sim = new QuantumSimulator())
{
    // Try initial values
    Result[] initials = new Result[] { Result.Zero, Result.One };
    foreach (Result initial in initials)
    {
        var res = BellTest.Run(sim, 1000, initial).Result;
        var (numZeros, numOnes, agree) = res;
        System.Console.WriteLine(
            $"Init:{initial,-4} 0s={numZeros,-4} 1s={numOnes,-4} agree={agree,-4}");
    }
}
System.Console.WriteLine("Press any key to continue...");
System.Console.ReadKey();
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image165.png)

And now we're ready to run the program. What we see is when the first qubit is initialized to Zero (i.e our input is |00⟩), the Hadamard gate puts it in a superposition, so that upon measurement it is |0⟩ fifty percent of the time, and |1⟩ fifty percent of the time – which you can see by the count of 0s and 1s. If the second bit was unaltered by the measurement of the first bit, it would remain |0⟩ and so would only agree 499 times.

However we can see that the second bit mirrored the value of the first bit exactly – it was |0⟩ (about) fifty percent of the time, and |1⟩ fifty percent of the time - resulting in the measurements agreeing 1000 times. This is exactly what we expect to see from Bell states!

Output Copy

```
Init:Zero 0s=499  1s=501  agree=1000
Init:One  0s=490  1s=510  agree=1000
```

(https://msdnshared.blob.core.windows.net/media/2018/02/image611.png)

And we're done! You've just completed your first quantum program and if you got this far that hopefully means you understood what you were doing. Time for a celebratory cup of tea.

(https://msdnshared.blob.core.windows.net/media/2018/02/image133.png)

What's next?

There's a lot of samples on GitHub which you can find here: https://github.com/Microsoft/Quantum/tree/master/Samples (https://github.com/Microsoft/Quantum/tree/master/Samples)

Next up in this blog series I'll be explaining the background of quantum teleportation and working through the code sample.

Also check out Anita's blog for a more in-depth look at quantum gates (Anita is amazing). https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/introduction-to-quantum-computing/ (https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/introduction-to-quantum-computing/)

Further resources

There were a few resources I found extremely helpful and I suggest you read if you want to go into more depth. The first is 'Quantum Information and Quantum Computation' by Nielsen and Chuang. The second is the SDK documentation by Microsoft: https://docs.microsoft.com/en-us/quantum/quantum-concepts-1-intro?view=qsharp-preview (https://docs.microsoft.com/en-us/quantum/quantum-concepts-1-intro?view=qsharp-preview)

Learn more at the Microsoft Quantum website: https://www.microsoft.com/en-us/quantum/ (https://www.microsoft.com/en-us/quantum/)

Download the Quantum Development Kit: (https://www.microsoft.com/en-us/quantum/development-kit)https://www.microsoft.com/en-us/quantum/development-kit (https://www.microsoft.com/en-us/quantum/development-kit)

Stay up to date with the Microsoft Quantum newsletter: (https://info.microsoft.com/Quantum-Computing-Newsletter-Signup.html) (https://info.microsoft.com/Quantum-Computing-Newsletter-Signup.html)https://info.microsoft.com/Quantum-Computing-Newsletter-Signup.html (https://info.microsoft.com/Quantum-Computing-Newsletter-Signup.html)

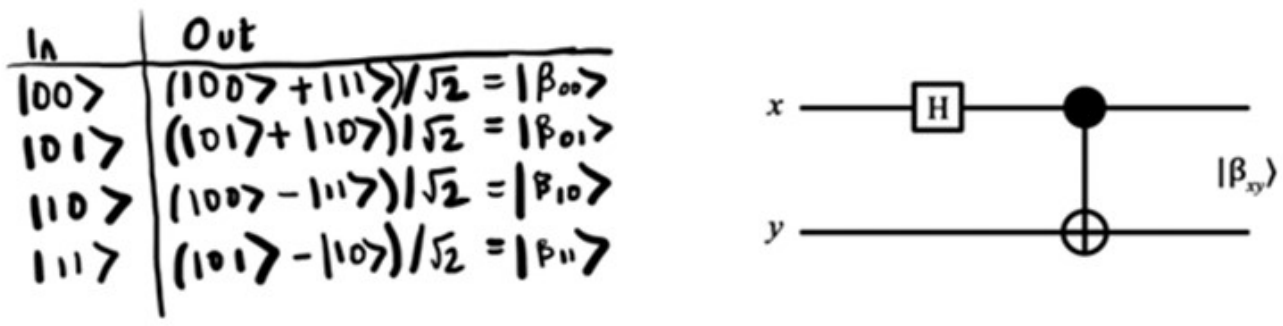
Read about the latest developments on the Microsoft Quantum blog: (https://cloudblogs.microsoft.com/quantum/)https://cloudblogs.microsoft.com/quantum/ (https://cloudblogs.microsoft.com/quantum/)

Twitter: https://twitter.com/MSFTQuantum (https://twitter.com/MSFTQuantum)

Facebook: https://www.facebook.com/MicrosoftQuantum/ (https://www.facebook.com/MicrosoftQuantum/)

I could make another blog post on other resources if readers would find that helpful (leave a comment!).

Appendix – Bell states



(https://msdnshared.blob.core.windows.net/media/2018/02/image134.png)

We can generate Bell states using a combination of a Hadamard gate followed by a CNOT. The Hadamard transforms the first qubit by putting it into a superposition, this is then used by the CNOT as a control to modify the target qubit. That is neatly described by the circuit diagram

Let’s begin with the first case, if the input is |00⟩. The first qubit |0⟩ passes through the Hadamard gate producing $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image135.png) , whilst the second qubit is unchanged. So we have:

$$|00\rangle \rightarrow \frac{(|0\rangle+|1\rangle)}{\sqrt{2}}|0\rangle = \frac{|00\rangle}{\sqrt{2}} + \frac{|10\rangle}{\sqrt{2}}$$

(https://msdnshared.blob.core.windows.net/media/2018/02/image136.png)

We then apply the CNOT gate which takes |00⟩→|00⟩ and |10⟩→|11⟩ so the state becomes:

$$|00\rangle \rightarrow \frac{|00\rangle}{\sqrt{2}} + \frac{|11\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image137.png)

Next the second case, if the input is |01⟩. The first qubit |0⟩ passes through the Hadamard gate producing $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image138.png) , whilst the second qubit is unchanged. So we have:

$$|01\rangle \rightarrow \frac{(|0\rangle+|1\rangle)}{\sqrt{2}}|1\rangle = \frac{|01\rangle}{\sqrt{2}} + \frac{|11\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image139.png)

We then apply the CNOT gate which takes |01⟩→|01⟩ and |11⟩→|10⟩ so the state becomes:

$$|01\rangle \rightarrow \frac{|01\rangle}{\sqrt{2}} + \frac{|10\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image140.png)

Then the third case, if the input is |10⟩. The first qubit |1⟩ passes through the Hadamard gate producing $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image141.png) , whilst the second qubit is unchanged. So we have:

$$|10\rangle \rightarrow \frac{(|0\rangle-|1\rangle)}{\sqrt{2}}|0\rangle = \frac{|00\rangle}{\sqrt{2}} - \frac{|10\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image142.png)

We then apply the CNOT gate which takes |00⟩→|00⟩ and |10⟩→|11⟩ so the state becomes:

$$|10\rangle \rightarrow \frac{|00\rangle}{\sqrt{2}} - \frac{|11\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image143.png)

And finally the fourth case, if the input is |11⟩. The first qubit |1⟩ passes through the Hadamard gate producing $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ (https://msdnshared.blob.core.windows.net/media/2018/02/image144.png) , whilst the second qubit is unchanged. So we have:

$$|11\rangle \rightarrow \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}|1\rangle = \frac{|01\rangle}{\sqrt{2}} - \frac{|11\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image145.png)

We then apply the CNOT gate which takes |01⟩→|01⟩ and |11⟩→|10⟩ so the state becomes:

$$|11\rangle \rightarrow \frac{|01\rangle}{\sqrt{2}} - \frac{|10\rangle}{\sqrt{2}}$$ (https://msdnshared.blob.core.windows.net/media/2018/02/image146.png)

Phew, all done. So, in summary, just as you can see in the table above, we have:

Tags Academic (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/academic/) Faculty (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/faculty/) learning (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/learning/) Q# (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/q/) Quantum (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/quantum/) Quantum Computing (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/quantum-computing/) Student (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/student/) Students (https://blogs.msdn.microsoft.com/uk_faculty_connection/tag/students/)

Comments (1)

Name *

Email *

Website

☐Save my name, email, and website in this browser for the next time I comment.

Post Comment

Wiretripper (https://social.msdn.microsoft.com/profile/Wiretripper)
September 4, 2018 at 1:33 pm (https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/a-beginners-guide-to-quantum-computing-and-q/#comment-117775)
An excellent overview! You should put this on Medium.

Reply (https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/06/a-beginners-guide-to-quantum-computing-and-q/?replytocom=117775#respond)