Q Sharp

Q# (pronounced as *Q sharp*) is a domain-specific programming language used for expressing quantum algorithms.[1] It was initially released to the public by Microsoft as part of the Quantum Development Kit.[2]

Q#	
Paradigm	multi-paradigm: quantum, functional, imperative
Designed by	Microsoft Research (quantum architectures and computation group; QuArC)
Developer	Microsoft
First appeared	December 11th, 2017
Typing discipline	static, strong
Platform	Common Language Infrastructure
License	MIT License
Filename extensions	.qs
Website	Microsoft Quantum (https://docs.microsoft.com/en-us/quantum) (GitHub (https://github.com/Microsoft/Quantum))
Influenced by	
C#, F#	

Contents

History

During a Microsoft Ignite Keynote on September 26, 2017, Microsoft announced that they were going to release a new programming language geared specifically towards quantum computers.[3] On December 11, 2017, Microsoft released Q# as a part of the Quantum Development Kit.[2]

Usage

Q# is available as a separately downloaded extension for Visual Studio,[4] but it can also be run as an independent tool from the Command line and/or Visual Studio Code. The Quantum Development Kit ships with a quantum simulator which is capable of running Q#.

In order to invoke the quantum simulator, another .NET programming language, usually C#, is used, which provides the (classical) input data for the simulator and reads the (classical) output data from the simulator.

Features

A primary feature of Q# is the ability to create and use qubits for algorithms. As a consequence, some of the most prominent features of Q# are the ability to entangle and introduce superpositioning to qubits via Controlled NOT gates and Hadamard gates, respectively, as well as Toffoli Gates, Pauli X, Y, Z Gate, and many more which are used for a variety of operations; see the list at the article on quantum logic gates.

The hardware stack that will eventually come together with Q# is expected to implement Qubits as topological qubits. The quantum simulator that is shipped with the Quantum Development Kit today is capable of processing up to 32 qubits on a user machine and up to 40 qubits on Azure.

Documentation and Resources

Currently, the resources available for Q# are scarce, but the official documentation is published: Microsoft Developer Network: Q# (https://docs.microsoft.com/en-us/quantum/?view=qsharp-preview). Microsoft Quantum Github repository (https://github.com/Microsoft/Quantum/) is also a large collection of sample programs implementing a variety of Quantum algorithms and their tests.

Microsoft has also hosted a Quantum Coding contest on Codeforces here: Microsoft Q# Coding Contest - Codeforces (https://web.archive.org/web/20181119064628/https://codeforces.com/msqs2018), and also provided related material to help answer the questions in the blog posts, plus the detailed solutions in the tutorials.

Microsoft hosts a set of learning exercises to help learn Q# on github: microsoft/QuantumKatas (https://github.com/Microsoft/QuantumKatas) with links to resources, and answers to the problems.

Syntax

Q# is syntactically related to both C# and F# yet also has some significant differences.

Similarities with C#

- Uses `namespace` for code isolation
- All statements end with a `;`
- Curly braces are used for statements of scope
- Single line comments are done using `//`
- Variable data types such as `Int` `Double` `String` and `Bool` are similar, although capitalised (and Int is 64-bit)[5]
- Qubits are allocated and disposed inside a `using` block.
- Lambda functions using the => operator.
- Results are returned using the `return` keyword.

Similarities with F#

- Variables are declared using either `let` or `mutable`[1]
- First-order functions
- Modules, which are imported using the `open` keyword
- The datatype is declared after the variable name
- The range operator `..`
- `for … in` loops

- Every operation/function has a return value, rather than `void`. Instead of `void`, an empty Tuple `()` is returned.
- Definition of record datatypes (using the `newtype` keyword, instead of `type`).

Differences

- Functions are declared using the `function` keyword
- Operations on the quantum computer are declared using the `operation` keyword
- Lack of multiline comments
- Asserts instead of throwing exceptions
- Documentation is written in Markdown instead of XML-based documentation tags

References

1. QuantumWriter. "The Q# Programming Language" (https://docs.microsoft.com/en-us/quantum/quantum-qr-intro?view=qsharp-preview). *docs.microsoft.com*. Retrieved 2017-12-11.
2. "Announcing the Microsoft Quantum Development Kit" (https://cloudblogs.microsoft.com/quantum/2017/12/11/announcing-microsoft-quantum-development-kit/). Retrieved 2017-12-11.
3. "Microsoft announces quantum computing programming language" (https://cloudblogs.microsoft.com/quantum/2017/09/26/microsoft-announces-quantum-computing-programming-language/). Retrieved 2017-12-14.
4. QuantumWriter. "Setting up the Q# development environment" (https://docs.microsoft.com/en-us/quantum/quantum-installconfig?view=qsharp-preview). *docs.microsoft.com*. Retrieved 2017-12-14.
5. "Types in Q# - Microsoft Quantum" (https://docs.microsoft.com/en-us/quantum/user-guide/language/types). *docs.microsoft.com*.

External links

- Official documentation (https://docs.microsoft.com/en-us/quantum/quantum-qr-intro?view=qsharp-preview)
- qsharp-language (https://github.com/microsoft/qsharp-language) on GitHub