Microsoft

Programming assignment 5
Debugging Grover's search algorithm

Mariia Mykhailova
Senior Software Engineer
Microsoft Quantum Systems

Step 1: Compilation errors

- Missing **open** statement for math functions

	Code	Description
⊗	QS5022	No identifier with the name "Round" exists.
⊗	QS5022	No identifier with the name "PI" exists.
⊗	QS5022	No identifier with the name "Sqrt" exists.

GroversSearch.qs;Index

```
1   namespace Quantum.Week5
2   {
3
4       open Microsoft.Quantum.Arrays;
5       open Microsoft.Quantum.Convert;
6       open Microsoft.Quantum.Measurement;
7       open Microsoft.Quantum.Intrinsic;
8       open Microsoft.Quantum.Canon;
9
```

GroversSearch.qs

```
1   namespace Quantum.Week5
2   {
3       open Microsoft.Quantum.Math;
4       open Microsoft.Quantum.Arrays;
5       open Microsoft.Quantum.Convert;
6       open Microsoft.Quantum.Measurement;
7       open Microsoft.Quantum.Intrinsic;
8       open Microsoft.Quantum.Canon;
9
```

- Cannot generate **`Adjoint`** automatically (within-apply block takes adjoint of its contents)

	Code	Description
⊗	QS6312	The called operation does not support the necessary functor(s) for the requested auto-generation of a specialization. Missing support for functor(s) Adjoint.

GroversSearch.qs;Index

```
15      operation Oracle_MarkedBitIsTrue (register: Qubit[],
16          let N = Length(register);
17          let selectBits = register[N/2 .. N];
18          within {
19              ApplyToEach(X, selectBits);
20          } apply {
21              Controlled Z(selectBits, target);
22          }
23      }
```

GroversSearch.qs

```
15      operation Oracle_MarkedBitIsTrue (register: Qubit[],
16          let N = Length(register);
17          let selectBits = register[N/2 .. N];
18          within {
19              ApplyToEachA(X, selectBits);
20          } apply {
21              Controlled Z(selectBits, target);
22          }
23      }
```

- Incorrect variable name

	Code	Description
⊗	QS5022	No identifier with the name "iter" exists.

GroversSearch.qs Driver.cs Diff - GroversSea

GroversSearch.qs;Index

```
38      operation GroversSearch_Loop (register : Qubit
39          let phaseOracle = OracleConverterImpl(orac
40          ApplyToEach(H, register);
41
42          for (i in 1 .. iter) {
43              // Apply the phase oracle
44              phaseOracle(register);
```

GroversSearch.qs

```
38      operation GroversSearch_Loop (register : Qubit[], oracle
39          let phaseOracle = OracleConverterImpl(oracle, _);
40          ApplyToEach(H, register);
41
42          for (i in 1 .. iterations) {
43              // Apply the phase oracle
44              phaseOracle(register);
```

- Incompatible types **`Int`** and **`Double`**: Q# doesn't have implicit type casting

	Code	Description
⊗	QS6210	The type of the given argument does not match the expected type. Got an argument of type Int, expecting one of type Double instead.
⊗	QS5008	The given arguments of type Int and Double do not have a common base type.
⊗	QS5008	The given arguments of type Int and Double do not have a common base type.

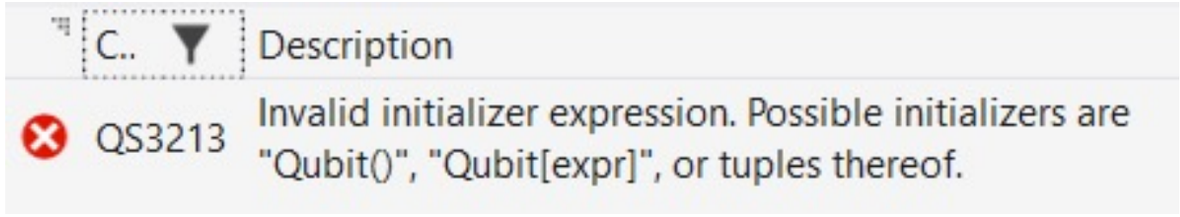
GroversSearch.qs;Index

```
55      operation GroversSearch_Main () : Unit {
56          let nQubits = 8;
57          let searchSpaceSize = 2 ^ nQubits;
58          let solutionsNumber = 2 ^ (nQubits/2);
59          let iter = Round(PI() / 4.0 *
60              Sqrt(searchSpaceSize * 1.0 / solutionsNumber));
61
```

GroversSearch.qs

```
55      operation GroversSearch_Main () : Unit {
56          let nQubits = 8;
57          let searchSpaceSize = 2 ^ nQubits;
58          let solutionsNumber = 2 ^ (nQubits/2);
59          let iter = Round(PI() / 4.0 *
60              Sqrt(IntAsDouble(searchSpaceSize) / IntAsDouble(solutionsNumber)));
61
```

- Incorrect syntax for qubit allocation



GroversSearch.qs;HEAD

```
57      mutable answer = new Bool[nQubits];
58      use (register, output) = (Qubit[nQubits], Qubit(1));
59      mutable correct = false;
```

GroversSearch.qs

```
57      mutable answer = new Bool[nQubits];
58      use (register, output) = (Qubit[nQubits], Qubit());
59      mutable correct = false;
```

Step 2: Runtime errors

- Incorrect range for subarray indices

  (array elements are indexed from 0 to (Length - 1), inclusive, and ranges of integers include their right bound)

```
operation Oracle_MarkedBitIsTrue (register: Qubit[], target: Qubit) : Unit {
    let N = Length(register);
    let selectBits = register[N/2 .. N];
    within {
        ApplyToEach(X, selectBits);
    } apply {
        Controlled Z(selectBits, target);
    }
}
```

Exception User-Unhandled

System.ArgumentOutOfRangeException: 'Specified argument was out of the range of valid values.'

GroversSearch.qs;Index

```
15  operation Oracle_MarkedBitIsTrue (register:
16      let N = Length(register);
17      let selectBits = register[N/2 .. N];
18      within {
19          ApplyToEach(X, selectBits);
20      } apply {
21          Controlled Z(selectBits, target);
22      }
23  }
```

GroversSearch.qs

```
15  operation Oracle_MarkedBitIsTrue (register: Qubit[],
16      let N = Length(register);
17      let selectBits = register[N/2 .. N - 1];
18      within {
19          ApplyToEach(X, selectBits);
20      } apply {
21          Controlled Z(selectBits, target);
22      }
23  }
```

- Released qubits have to be in **|0⟩** state

  (here the ancilla qubit is in **|−⟩** state instead)

```
operation OracleConverterImpl (markingOracle : ((Qubit[],
    using (target = Qubit()) {
        // Put the target into the |-⟩ state
        X(target);
        H(target);
        // Apply the marking oracle; since the target is
        // flipping the target if the register satisfies
        markingOracle(register, target);
```

Exception User-Unhandled

Microsoft.Quantum.Simulation.Simulators.Exceptions.ReleasedQubitsAreNotInZeroState: 'Released qubits are not in zero state.'

GroversSearch.qs;HEAD

```
23      operation OracleConverterImpl (markingOracle :
24          use target = Qubit();
25          // Put the target into the |-⟩ state
26          X(target);
27          H(target);
28          // Apply the marking oracle; since the targ
29          // flipping the target if the register sati
30          markingOracle(register, target);
31      }
```

GroversSearch.qs

```
23      operation OracleConverterImpl (markingOracle :
24          use target = Qubit();
25          // Put the target into the |-⟩ state
26          X(target);
27          H(target);
28          // Apply the marking oracle; since the targ
29          // flipping the target if the register sati
30          markingOracle(register, target);
31          H(target);
32          X(target);
33      }
```

Step 3: Logic/runtime errors

- Released qubits have to be in $|0\rangle$ state (again!)

  This time it's an earlier logic error manifesting

  Marking oracles use `Controlled X` to write the result to target, not `Controlled Z`

```
operation Oracle_MarkedBitIsTrue (register: Qubit[], target: Qubit)
    let N = Length(register);
    let selectBits = register[N/2 .. N - 1];
    within {
        ApplyToEachA(X, selectBits);
    } apply {
        Controlled Z(selectBits, target);
    }
}

operation OracleConverterImpl (markingOracle : ((Qubit[], Qubit) =>
    use target = Qubit();
```

Exception User-Unhandled

Microsoft.Quantum.Simulation.Simulators.Exceptions.ReleasedQubitsAreNotInZeroState: 'Released qubits are not in zero state.'

View Details | Copy Details | Start Live Share session...

▷ Exception Settings

```
e |-) state
racle cond
(target);
}
```

- Does not print the actual solution

```
{answer}
```

Need to use interpolated string instead of a regular one

GroversSearch.qs;Index

```
75              }
76              Message("{answer}");
77          }
78      }
79
```

GroversSearch.qs

```
75              }
76              Message($"{answer}");
77          }
78      }
79
```

- Sometimes outputs incorrect result!

```
[True,True,False,True,True,True,False,False]
```

Use correct basis to measure when checking for correctness

GroversSearch.qs;Index

```
64    repeat {
65        GroversSearch_Loop(register, Oracle_MarkedBitIsTrue, iter);
66        let res = MultiM(register);
67        // Check whether the result is correct
68        Oracle_MarkedBitIsTrue(register, output);
69        if (MResetX(output) == One) {
70            set correct = true;
71            set answer = ResultArrayAsBoolArray(res);
72        }
73        ResetAll(register);
74    } until (correct);
```

GroversSearch.qs

```
64    repeat {
65        GroversSearch_Loop(register, Oracle_MarkedBitIsTrue, iter);
66        let res = MultiM(register);
67        // Check whether the result is correct
68        Oracle_MarkedBitIsTrue(register, output);
69        if (MResetZ(output) == One) {
70            set correct = true;
71            set answer = ResultArrayAsBoolArray(res);
72        }
73        ResetAll(register);
74    } until (correct);
```