CLIQ Language Manual

1 Introduction

CLIQ is a quantum circuit simulator and DSL (domain-specific language) for describing quantum programs. It supports gate application, parametric gates, measurement, oracles, and structured control (for/if/while), with a macro system and scoped definitions. This manual documents the language syntax and behavior as implemented in the codebase.

Conventions

- Square brackets denote qubit lists, e.g., [0] or [0,1] . Qubits are 0-based. - Inline comments use # and extend to end of line. - Identifiers match [A−Za−z_][A−Za−z0−9_]∗.

2 Getting Started

A program is a sequence of lines. Empty lines and comment lines are ignored.

2.1 DEFINE

DEFINE NAME = VALUE
DEFINE "NAME WITH SPACES" = VALUE

Defines are token-aware and can be used anywhere later in the file, including inside control structures and macros. Values are stringly stored but many parts of the parser will resolve identifiers and evaluate simple arithmetic when needed.

Common values: - Numbers: 4, 0, etc. - Ranges: range(n), range(start ,end), range(start ,end,step) (step is supported in for-loop iterables; see §5.1). - CSVs: 0,1,2,3 (used as target lists when indexed).

2.2 Ranges and Indexing

You can form ranges with range (...) and extract specific indices from *defined* names using an indexing suffix:

```
DEFINE NUM_QUBITS = 4
DEFINE CIRCUIT = range(NUM_QUBITS)
X [CIRCUIT[end]]    # selects the last element -> X [3]
```

Index forms supported when indexing a defined name: - [k] for zero-based integer k\ge 0 - [−1] or [end] for the last index - Negative indices: [−2] means second-from-last, etc. Bounds are validated; out-of-bounds raises a parser warning/error.

Note: range step (range(a,b,step)) is supported in for-loop iterables, but not inside bracket qubit lists. In bracket lists you may use range(n) or range(a,b), or explicit CSV.

2.3 Arithmetic in Brackets

Arithmetic inside brackets is evaluated:

```
X [4 - 1]          # becomes X [3]
U [2 + 3]
```

3 Gates

General gate form:

```
NAME [controls] [targets]
NAME [targets]          # controls omitted
```

Examples:

```
H [] [0]
X [0] [1]
SWAP [1,2]              # 2-target special gate
```

3.1 Parametric Gates

Supported parametric gates: **PH**, **RPH**, **RX**, **RY**, **RZ**.

```
RX(pi/4) [] [0]
RZ(2*pi) [1] [0]
Ph(pi/8) [] [2]
```

Angle expressions support numbers, fractions, and π: pi/4, 2*pi, −pi/3, pi*2/3, or decimals.

3.2 Measurement

```
! [targets]
```

Example: [0,1] ! measures qubits 0 and 1. Measurement results are recorded for use in conditionals/while.

4 Oracle

```
ORACLE [input_qubits] [output_qubit] [marked_states]
```

Marked states can be decimal (e.g., [0,1,5]) or binary with a b prefix (e.g., b[111,001,010]). The implementation decomposes oracles into CNOT/X gates.

5 Control Structures

5.1 for

Two forms are supported:

1) Legacy count-based:

```
for [N]:
  ...
end
```

Expands the body N times, with an informational note per iteration.

2) Variable-based with iterable and substitution:

```
for i [range(0, 4)]:
  X [] [i]
end
```

Valid iterables: range(n), range(a,b), range(a,b,step), CSV (e.g., 0,2,3), or a DEFINE name that resolves to one of these. The loop variable is textually substituted in the body at token boundaries.

5.2 if

Conditionals evaluate boolean expressions over measurement results. Syntax:

```
if [q] == 1 and ([r] != 0 or [s] == 1):
  ...
end
```

Operators: ==, !=; logical connectors: and, or (or &&, ||). Parentheses are supported and respect precedence. If a qubit has not been measured, its value defaults to 0 for evaluation.

5.3 while

Same boolean syntax as if. A safety cap limits iterations (default: 10,000) to prevent infinite loops.

```
while [0] == 0:
  H [] [0]
  ! [0]
end
```

5.4 WITH (Scoped DEFINE overrides)

Temporarily override or introduce DEFINEs for a block:

```
WITH A=1, B=range(4):
  X [] [B[end]]  # uses B from the WITH scope
end
```

Overrides are pushed before the block and restored afterwards.

6 Macros

Define and invoke reusable snippets with parameters.

```
MACRO ApplyPair [a, b]:
  H [] [a]
  CNOT [a] [b]
end

ApplyPair [0,1]
```

- Parameters are substituted token-wise. - Invocation supports positional [0,1] or named form [a=0,b=1]. Nested control structures inside macros are supported.

7 Language Semantics

7.1 Qubit Counting

The simulator infers the required number of qubits from all gate targets/controls across the full program, including those inside conditionals and loops. The total qubit count is max _index + 1, with a minimum of 1.

7.2 Definition Resolution

The parser repeatedly applies DEFINE substitutions with a capped number of passes. When indexing a defined name that holds a range (e.g., range(NUM_QUBITS)), the value is resolved so that constructs like [end] and negative indices work as expected.

7.3 Bracket Arithmetic

Any arithmetic expression like [4−1] is evaluated (addition and subtraction). More complex arithmetic may be rejected if it cannot be reduced to integers.

8 Examples

8.1 Bell State

```
H [] [0]
X [0] [1]
! [0,1]
```

8.2 Iterating Over a Range

```
DEFINE N = 4
for i [range(N)]:
  H [] [i]
end
```

8.3 Conditional on Measurement

```
H [] [0]
! [0]
if [0] == 1:
  X [] [1]
end
```

8.4 Oracle (decimal and binary)

```
# Mark states 1 and 3 on inputs [0,1], output [2]
ORACLE [0,1] [2] [1,3]
# Binary form marking 111 and 010 on inputs [0,1,2], output [3]
ORACLE [0,1,2] [3] b[111,010]
```

9 Runtime Interfaces

9.1 CLI

Interactive commands: `forward/backward/reset/run/frun/rrun/draw/show`. The state vector and marginal probabilities can be printed.

10 Platform Notes

- Windows: run the CLI with the `.jar` artifact. - Linux: if an AppImage is available, use it; otherwise the `.jar` works with any recent JRE.

11 Error Handling

The parser emits warnings for unresolvable indices, malformed ranges, and unmatched control structure `end`s. Many operations fail fast with an error message including a line number.

12 Appendix

Reserved Gate Names

Standard gates like `X,Y,Z,H,SWAP`, and parametric `PH,RPH,RX,RY,RZ`. Custom matrices can also be defined via `MATRIX: name = [ ... ]` and then referenced by `name [C] [T]`.