OpenQASM 3 (legible) grammar

```
      <program>  ::=  <globalstmt|stmt>*

   <globalstmt>  ::=  <subroutinedef>
                   |  <externdef>
                   |  <gatedef>
                   |  <calibration>
                   |  <qdecl>
                   |  #pragma { <stmt>* }
<subroutinedef>  ::=  def <id>(<arg...>?)  { <stmt>* <returnstmt>?  }
                   |  def <id>(<arg...>?)  -> <ctype> { <stmt>* <returnstmt>?  }
    <externdef>  ::=  extern <id>(<ctype...>?);
                   |  extern <id>(<ctype...>?)  -> <ctype>;
      <gatedef>  ::=  gate <id> <id...> <qblock>
                   |  gate <id>(<id...>?)  <id...> <qblock>
  <calibration>  ::=  defcalgrammar "openpulse";
                   |  defcalgrammar <string>;
                   |  defcal <id> <id...> { .* }
                   |  defcal <id>(<carg...>?)  <id...> { .* }
                   |  defcal <id>(<exp...>?)  <id...> { .* }
                   |  defcal <id> <id...> -> <ctype> { .* }
                   |  defcal <id>(<carg...>?)  <id...> -> <ctype> { .* }
                   |  defcal <id>(<exp...>?)  <id...> -> <ctype> { .* }
        <qdecl>  ::=  qreg <id>; | qreg <id>[<exp>];
                   |  qubit <id>; | qubit[<exp>] <id>;

         <stmt>  ::=  <exp>;
                   |  <id> <assignop> <exp>; | <id>[<exp>] <assignop> <exp>;
                   |  measure <indexid> -> <indexid>;
                   |  <indexid> = measure <indexid>;
                   |  <classicdecl>
                   |  if (<exp>) <prgmblock>
                   |  if (<exp>) <prgmblock> else <prgmblock>
                   |  for <id> in <setdecl> <prgmblock>;
                   |  while (<exp>) <prgmblock>;
                   |  end;
                   |  let <id> = <indexid>;
                   |  <qstmt>

        <qstmt>  ::=  <gatemodifier>* <id> <indexid...>;
                   |  <gatemodifier>* <id>(<exp...>) <indexid...>;
                   |  <gatemodifier>* gphase(<exp>) <indexid...>?;
                   |  measure <indexid>;
                   |  reset <indexid...>;
                   |  barrier <indexid...>?;
                   |  <timestmt>
```

```
<timestmt>  ::=  delay[<exp>] <indexid...>;
               | delay(<exp...>?)[<exp>] <indexid...>;
               | rotary[<exp>] <indexid...>;
               | rotary(<exp...>?)[<exp>] <indexid...>;
               | box <qblock>
               | box[<exp>] <qblock>

<prgmblock>  ::=  <stmt>
               | <control>
               | { <stmt|control>* }
<control>    ::=  break; | continue;
               | <returnstmt>
<returnstmt> ::=  return;
               | return <exp>;
               | return measure <indexid>;

<qblock>     ::=  { <qstmt|qloop>* }
<qloop>      ::=  for <id> in <setdecl> <qstmt>
               | for <id> in <setdecl> { <qstmt>* }
               | while (<exp>) <qstmt>
               | while (<exp>) { <qstmt>* }

<arg>        ::=  <carg> | <qarg>
<carg>       ::=  <singledesignatortype>[<exp>] <id>
               | <nodesignatortype> <id>
               | creg <id> | creg <id>[<exp>]
               | bit <id> | bit[<exp>] <id>
               | complex[<singledesignatortype>[<exp>]] <id>
<qarg>       ::=  qreg <id> | qreg <id>[<exp>]
               | qubit <id> | qubit[<exp>] <id>

<ctype>      ::=  <singledesignatortype>[<exp>]
               | <nodesignatortype>
               | bit | bit[<exp>] | creg | creg[<exp>]
               | complex[<singledesignatortype>[<exp>]]

<singledesignatortype>  ::=  int | uint | float | angle
<nodesignatortype>      ::=  bool | duration | stretch

<indexid>    ::=  <id>[<exp>?:<exp>?]
               | <id>[<exp>?:<exp>?:<exp>]
               | <id>
               | <id>[<exp...>]
               | <indexid>||<indexid>
<setdecl>    ::=  { <exp...> }
               | [<exp>?:<exp>?] | [<exp>?:<exp>?:<exp>]
               | <id>

<equalsexp>  ::=  = <exp>
<classicdecl>  ::=  <singledesignatortype>[<exp>] <id> <equalsexp>?;
               | <nodesignatortype> <id> <equalsexp>?;
               | creg <id> <equalsexp>?; | creg <id>[<exp>] <equalsexp>?;
               | bit <id> <equalsexp>?; | bit[<exp>] <id> <equalsexp>?;
```

```
                  |  complex[<singledesignatortype>[<exp>]] <id> <equalsexp>?;
                  |  const <ctype> <id> <equalsexp>;

          <exp>  ::=  <constant> | <int> | <real> | <imag>
                  |  <bool> | <id> | <string>
                  |  <math>(<exp...>)
                  |  <ctype>(<exp...>)
                  |  <id>(<exp...>?)
                  |  <time> | durationof(<id>)
                  |  durationof(<qblock>)
                  |  (<exp>)
                  |  <exp>[<exp>]
                  |  <exp> <b-op> <exp>
                  |  <u-op> <exp>

         <math>  ::=  arcsin | sin | arccos | cos | arctan | tan
                  |  exp | ln | sqrt | rotl | rotr | popcount
         <b-op>  ::=  || | && | | | ^ | & | == | != | > | < | >= | <=
                  |  << | >> | + | - | * | / | % | **
         <u-op>  ::=  ~ | ! | -

    <assignop>  ::=  = | += | -= | *= | /= | &= | |= | ~= | ^= | <<= | >>= | %= | **=

<gatemodifier>  ::=  inv @ | pow(<exp>) @
                  |  ctrl @ | ctrl(<exp>) @
                  |  negctrl @ | negctrl(<exp>) @

          <uni>  ::=  [\p{Lu}\p{Ll}\p{Lt}\p{Lm}\p{Lo}\p{Nl}]
           <id>  ::=  (_|$|<uni>|[A-Za-z])(_|$|<uni>|[A-Za-z]|[0-9])*
          <int>  ::=  [0-9]+
         <real>  ::=  ([0-9]+|[0-9]+.[0-9]*)([eE][+-]?[0-9]+)?
         <imag>  ::=  (<int>|<real>)im
         <bool>  ::=  true|false
         <time>  ::=  (<int>|<real>)(dt|ns|us|µs|ms|s)
       <string>  ::=  "[^"\r\t\n]*" | '[^'\r\t\n]*'
     <constant>  ::=  pi | π | tau | τ | euler | ε

      <comment>  ::=  // ...
                  |  /* ...  */
```