Setanta grammar specification

Below is the full specification for the *Setanta* syntax, as used by *tsPEG* to generate it's input parser

```
1  ---
2  import { Environment } from "./env";
3  import { callFunc, idxList, Value } from "./values";
4  import { unescapeChars } from "./litreacha";
5  import * as Asserts from "./asserts";
6  import * as Checks from "./checks";
7  import { orBinOp, orQuickBinOp, andBinOp, andQuickBinOp,
8      binOpEvalFn, binOpQuickEvalFn } from "./binops";
9  import { objLookupsEval, postfixArgsEval, csArgsEval, prefEval, EvalFn } from "./
       evals";
10 import { qEvalToEval } from "./evals";
11 import * as Quick from "./quickevals";
12 ---
13 Program      := stmts=AsgnStmt* _
14 AsgnStmt    := IfStmt
15              | BlockStmt
16              | NuairStmt
17              | LeStmt
18              | CCStmt
19              | BrisStmt
20              | CtlchStmt
21              | GniomhStmt
22              | ToradhStmt
23              | AssgnStmt
24              | DefnStmt
25              | Expr
26 NonAsgnStmt := IfStmt
27              | NuairStmt
28              | LeStmt
29              | CCStmt
30              | BrisStmt
31              | ToradhStmt
32              | BlockStmt
33              | AssgnStmt
34              | Expr
35 IfStmt       := _ 'm[áa]' \&gap expr=Expr \&gap stmt=NonAsgnStmt elsebranch={_ 'n[oó]
       ' \&gap stmt=NonAsgnStmt}?
36 BlockStmt    := _ '{' blk=AsgnStmt* _ '}'
37 NuairStmt    := _ 'nuair-a' expr=Expr \&gap stmt=NonAsgnStmt
38 LeStmt       := _ 'le' \&gap id=ID _ 'idir' _ '\('strt=Expr _ ',' end=Expr step={_ ',
       ' step=Expr}? _ '\)' stmt=NonAsgnStmt
39 DefnStmt     := _ id=ID _ ':=' _ expr=Expr
40 AssgnStmt    := _ lhs=Postfix _ op=AsgnOp _ expr=Expr
41 GniomhStmt   := _ 'gn[íi]omh' \&gap id=ID _ '\(' args=CSIDs? _ '\)' _ '{'
42     stmts=AsgnStmt*
43 _ '}'
44 CtlchStmt    := _ 'creatlach' \&gap id=ID tuis={_ 'ó' id=ID}? _ '{'
45     gniomhs=GniomhStmt*
46 _ '}'
47 BrisStmt     := _ 'bris'
```

```
CCStmt      := _ 'chun-cinn'
ToradhStmt  := _ 'toradh' \&gap exp=Expr?
Expr        := And
And         := head=Or tail={_ '\\&' trm=Or}*
               .evalfn = EvalFn { return andBinOp(this); }
               .qeval = Quick.MaybeEv { return andQuickBinOp(this); }
Or          := head=Eq tail={_ '\|' trm=Eq}*
               .evalfn = EvalFn { return orBinOp(this) }
               .qeval = Quick.MaybeEv { return orQuickBinOp(this); }
Eq          := head=Comp tail={_ op='[!=]=' trm=Comp}*
               .evalfn = EvalFn { return binOpEvalFn(this) }
               .qeval = Quick.MaybeEv { return binOpQuickEvalFn(this); }
Comp        := head=Sum tail={_ op=Compare trm=Sum}*
               .evalfn = EvalFn { return binOpEvalFn(this) }
               .qeval = Quick.MaybeEv { return binOpQuickEvalFn(this); }
Sum         := head=Product tail={_ op=PlusMinus trm=Product}*
               .evalfn = EvalFn { return binOpEvalFn(this) }
               .qeval = Quick.MaybeEv { return binOpQuickEvalFn(this); }
Product     := head=Prefix tail={_ op=MulDiv trm=Prefix}*
               .evalfn = EvalFn { return binOpEvalFn(this); }
               .qeval = Quick.MaybeEv { return binOpQuickEvalFn(this); }
Prefix      := _ op='-|!'? pf=Postfix
               .evalfn = EvalFn { return prefEval(this); }
               .qeval = Quick.MaybeEv { return Quick.qPrefEval(this); }
Postfix     := at=ObjLookups ops=PostOp*
               .evalfn = EvalFn { return postfixArgsEval(this); }
               .qeval = Quick.MaybeEv { return Quick.qPostfixArgsEval(this); }
ObjLookups  := _ attrs={id=ID '@' !wspace}* root=Atom
               .evalfn = EvalFn { return objLookupsEval(this); }
               .qeval = Quick.MaybeEv { return Quick.qObjLookupsEval(this); }
PostOp      := '\(' args=CSArgs? _ '\)' | '\[' expr=Expr '\]'
Atom        :=  _ '\(' trm=Expr '\)'
               .evalfn = EvalFn { return (env: Environment) => this.trm.evalfn(env);
                   }
               .qeval = Quick.MaybeEv {
                    const childF = this.trm.qeval;
                    return childF === null ? null : childF.bind(this.trm);
               }
             | ID
             | Litreacha
             | Int
             | Bool
             | Neamhni
             | ListLit
LSpec       := id=ID ops=PostOp*
ListLit     := _ '\[' els=CSArgs? _ '\]'
               .evalfn = EvalFn {
                   return (env: Environment) => this.els ? this.els.evalfn(env) :
                       Promise.resolve([]);
               }
               .qeval = Quick.MaybeEv { return Quick.qListLitEval(this); }
CSArgs      := head=Expr tail={_ ',' exp=Expr}*
               .evalfn = (env:Environment)=>Promise<Value[]> { return csArgsEval(
                   this); }
               .qeval = ((env:Environment)=>Value[])|null { return Quick.qCSArgsEval
```

```
                    (this); }
CSIDs       := head=ID tail={_ ',' id=ID}*
ID          := _ !{Keyword gap} id='[a-zA-Z_áéíóúÁÉÍÓÚ]+'
               .evalfn = EvalFn { return qEvalToEval(Quick.qIdEval(this.id)); }
               .qeval = Quick.EvalFn { return Quick.qIdEval(this.id); }
Bool        := _ bool='f[ií]or|br[eé]ag'
               .evalfn = EvalFn { return qEvalToEval(Quick.qBoolEval(this.bool)); }
               .qeval = Quick.EvalFn { return Quick.qBoolEval(this.bool); }
Neamhni     := _ 'neamhn[ií]'
               .evalfn = EvalFn { return () => Promise.resolve(null); }
               .qeval = Quick.EvalFn { return () => null; }
Int         := _ int='-?[0-9]+(?:\.[0-9]+)?'
               .evalfn = EvalFn { return qEvalToEval(Quick.qIntEval(this.int)); }
               .qeval = Quick.EvalFn { return Quick.qIntEval(this.int); }
Litreacha   := _ '\'' val='([^\'\\]|\\.)*' '\''
               .evalfn = EvalFn { return qEvalToEval(Quick.qLitreachaEval(this.val))
                   ; }
               .qeval = Quick.EvalFn { return Quick.qLitreachaEval(this.val); }
_           := wspace*
wspace      := '(?:\s|>--(?:(?!--<).)*(--<|\n|$))'
gap         := { wspace | '[^a-zA-Z0-9áéíóúÁÉÍÓÚ]' }+ | '$'
PlusMinus   := '\+|-'
AsgnOp      := '=|\+=|\*=|-=|%=|\/='
MulDiv      := '\*|\/\/|%|\/'
Compare     := '<=|>=|<|>'
Keyword     := 'm[áa]' | 'n[oó]' | 'nuair-a' | 'f[ií]or|br[eé]ag'
    | 'gn[ií]omh' | 'chun-cinn' | 'neamhn[ií]' | 'toradh' | 'creatlach'
```