Parsing for a Scripting Language (or any other sort)

---

grammar:

<while-stmt> → while **(** <expr> **)** <stmt>

parsing scheme:

```
node * parse_while_stmt()
{ call lexical analyser
  error if token is not reservedword while
  call lexical analyser
  error if token is not open paren
  p1 = parse_expr();
  call lexical analyser
  error if token is not close paren
  p2 = parse_stmt();
  return make_node(nt_while_stmt, p1, p2); }
```

---

grammar:

<assign-stmt> → <variable> **=** <expr> **;**

parsing scheme:

```
node * parse_variable()
{ call lexical analyser
  error if token is not identifier
  symbol * s1 = symbol table entry from the token
  return make_node_s(nt_variable, s1); }

node * parse_assign_stmt()
{ p1 = parse_variable();
  call lexical analyser
  error if token is not equal sign
  p2 = parse_expr();
  call lexical analyser
  error if token is not semicolon
  return make_node(nt_assign_stmt, p1, p2); }
```

---

grammar:

<block> → **{** <stmt-list> **}**

parsing scheme:

```
node * parse_print_stmt()
{ call lexical analyser
  error if token is not open curly
  p1 = parse_stmt_list();
  call lexical analyser
  error if token is not close curly
  return make_node(nt_block, p1); }
```

---

grammar:

```
<stmt-list> → Ø
            | <stmt> <stmt-list>
```

parsing scheme:

```
node * parse_stmt_stmt()
{ call lexical analyser
  put back (un-read) token
  if token is close curly*
    return NULL;
  p1 = parse_stmt();
  p2 = parse_expr_list();
  return make_node(nt_expr_list, p1, p2); }
```

* close-curly is the only thing that can ever follow a <stmt-list>, which is lucky

---

grammar:

<print-stmt> → print <expr-list> **;**

parsing scheme:

```
node * parse_print_stmt()
{ call lexical analyser
  error if token is not reservedword print
  p1 = parse_expr_list();
  call lexical analyser
  error if token is not semicolon
  return make_node(nt_print_stmt, p1); }
```

---

grammar:

```
<expr-list> → Ø
            | <expr> , <expr-list>
```

parsing scheme:

```
node * parse_print_stmt()
{ call lexical analyser
  put back (un-read) token
  if (token is in follow set* for <expr-list>)
    return NULL;
  p1 = parse_expr();
  call lexical analyser
  error if token is not comma
  p2 = parse_expr_list();
  return make_node(nt_expr_list, p1, p2); }
```

* Currently the follow set for <expr-list> only includes the semi-colon, but that will probably change

---

grammar:

```
<stmt> → <while_stmt>
       | <assign_stmt>
       | <print_stmt>
       | <block>
```

parsing scheme:

```
node * parse_stmt()
{ call lexical analyser
  put back (unread) token
  if token is reservedword while
    return parse_while_stmt();
  else if token is reservedword while
    return parse_while_stmt();
  else if token is identifier
    return parse_assign_stmt();
  else if token is reservedword print
    return parse_print_stmt();
  else if token is open_curly
    return parse_block();
  else
    error }
```

---

consider this grammar:

```
<if-stmt> → if ( <expr> ) <stmt> { else <stmt> }
```