Parser for Compiler in `mosmlyac`

Franklin Chen

July 28, 1996

This is a parser for the project developed in [**?**], done in `mosmlyac` for Moscow ML rather than in `yacc` for C.

The specification file has a well-defined format:

?? ⟨* ??⟩≡
```
%{
⟨Header ??⟩
%}
⟨Declarations ??⟩
%%
⟨Rules ??⟩
%%
⟨Trailer ??⟩
```

1 Header

We prefer not to `open` modules unnecessarily.

?? ⟨*Header* ??⟩≡ (? 0—1)
```
(* Empty *)
```

2 Declarations

?? ⟨*Declarations* ??⟩≡ (? 0—1)
```
⟨Tokens ??⟩
⟨Precedence ??⟩
⟨Entry points ??⟩
⟨Other nonterminal types ??⟩
```

2.1 Tokens

These are the only tokens with attributes.

?? ⟨*Tokens* ??⟩≡ (? 0—1) ??▷

```
%token <string> Identifier
%token <int> Constant
```

Defines:
Constant, never used.
Identifier, never used.

The rest of the tokens do not need attributes.

?? ⟨ *Tokens* ??⟩+≡ (? 0—1) ◁?? ??▷

```
%token INT
%token IF
%token ELSE
%token WHILE
%token BREAK
%token CONTINUE
%token RETURN
%token SEMI
%token LPAREN
%token RPAREN
%token LBRACE
%token RBRACE
%token PLUS
%token MINUS
%token TIMES
%token DIVIDE
%token REM
%token GT
%token LT
%token GE
%token LE
%token EQ
%token NE
%token AMP
%token CARET
%token BAR
%token ASSIGN
%token PE
%token ME
%token TE
%token DE
%token RE
%token PP
%token MM
%token COMMA
```

Defines:
AMP, never used.
ASSIGN, never used.
BAR, never used.
BREAK, never used.
CARET, never used.
COMMA, never used.
CONTINUE, never used.
DE, never used.
DIVIDE, never used.

ELSE, never used.
EQ, never used.
GE, never used.
GT, never used.
IF, never used.
INT, never used.
LBRACE, never used.
LE, never used.
LPAREN, never used.
LT, never used.
ME, never used.
MINUS, never used.
MM, never used.
NE, never used.
PE, never used.
PLUS, never used.
PP, never used.
RBRACE, never used.
RE, never used.
REM, never used.
RETURN, never used.
RPAREN, never used.
SEMI, never used.
TE, never used.
TIMES, never used.
WHILE, never used.

The usual end-of-file token.

?? ⟨ *Tokens* ?? ⟩+≡ (? 0—1) ◁ ??

```
%token EOF
```

Defines:
EOF, never used.

Precedences follow, from lowest to highest.

?? ⟨*Precedence* ??⟩≡ (? 0—1)

```
%right ASSIGN PE ME TE DE RE
%left BAR
%left CARET
%left AMP
%left EQ NE
%left LT GT GE LE
%left PLUS MINUS
%left TIMES DIVIDE REM
%right PP MM
```

Defines:
AMP, never used.
ASSIGN, never used.
BAR, never used.
CARET, never used.
DE, never used.
DIVIDE, never used.
EQ, never used.
GE, never used.
GT, never used.
LE, never used.
LT, never used.
ME, never used.
MINUS, never used.
MM, never used.
NE, never used.
PE, never used.
PLUS, never used.
PP, never used.
RE, never used.
REM, never used.
TE, never used.
TIMES, never used.

2.2 Nonterminals

We only have one entry point.

?? ⟨*Entry points* ??⟩≡ (? 0—1)

```
%start program
%type <Format.action> program
```

Uses program.

Note the pervasive passing around of `Format.action`.

?? ⟨*Other nonterminal types* ??⟩≡ (? 0—1)

```
%type <Format.action> definitions
%type <Format.action> definition
%type <Format.action> function_definition
%type <Format.action> optional_parameter_list
%type <Format.action> parameter_list
%type <Format.action> parameter_declarations
%type <Format.action> parameter_declaration
%type <Format.action> parameter_declarator_list
%type <Format.action> compound_statement
%type <Format.action> declarations
%type <Format.action> declaration
%type <Format.action> declarator_list
%type <Format.action> statements
%type <Format.action> statement
%type <Format.action> if_prefix
%type <Format.action> loop_prefix
%type <Format.action> expression
%type <Format.action> binary
%type <Format.action> optional_argument_list
%type <Format.action> argument_list
⟨Dummy nonterminal types ??⟩
```

Uses `argument_list`, `binary`, `compound_statement`, `declaration`, `declarations`, `declarator_list`, `definition`, `definitions`, `expression`, `function_definition`, `if_prefix`, `loop_prefix`, `optional_argument_list`, `optional_parameter_list`, `parameter_declaration`, `parameter_declarations`, `parameter_declarator_list`, `parameter_list`, `statement`, and `statements`.

3 Rules

Note that various synthesized attributes are now functions, in order to simulate the inherited formatter attribute we pass around to do pretty-printing.

?? ⟨*Rules* ??⟩≡ (? 0—1)

```
⟨Main entry point ??⟩
⟨Other rules ??⟩
```

?? ⟨*Main entry point* ??⟩≡ (? 0—1)

```
program
        : definitions EOF { $1 }
```

Defines:
`program`, never used.
Uses `definitions` and `EOF`.

We will expand each of these later.

?? ⟨*Other rules* ??⟩≡ (? 0—1)
⟨definitions ??⟩
⟨definition ??⟩
⟨function_definition ??⟩
⟨optional_parameter_list ??⟩
⟨parameter_list ??⟩
⟨parameter_declarations ??⟩
⟨parameter_declaration ??⟩
⟨parameter_declarator_list ??⟩
⟨compound_statement ??⟩
⟨declarations ??⟩
⟨declaration ??⟩
⟨declarator_list ??⟩
⟨statements ??⟩
⟨statement ??⟩
⟨if_prefix ??⟩
⟨loop_prefix ??⟩
⟨expression ??⟩
⟨binary ??⟩
⟨optional_argument_list ??⟩
⟨argument_list ??⟩
⟨*Rules for dummy nonterminals* ??⟩

Use function composition to build up a composite action that will be executed.

?? ⟨definitions ??⟩≡ (? 0—1)
```
definitions
        : definition { $1 }
        | definitions definition { $2 o $1 }
```
Defines:
definitions, never used.
Uses definition.

We replace the former INT with a simple nonterminal n_int in order to associate actions with tokens.

?? ⟨definition ??⟩≡ (? 0—1)
```
definition
        : function_definition { $1 }
        | n_int function_definition { $2 o $1 }
        | declaration { $1 }
```
Defines:
definition, never used.
Uses declaration, function_definition, and n_int.

Note that function_definition_1 and function_definition_2 had to be introduced because unlike yacc, mosmlyac does not support actions within rules.

?? ⟨function_definition ??⟩≡ (? 0—1)

```
function_definition
        : n_identifier n_lp optional_parameter_list n_rp
          function_definition_1
          parameter_declarations
          function_definition_2
          compound_statement { (Format.nl Format.AT) o
                               $8 o $7 o $6 o $5 o $4 o $3 o $2 o $1 }

function_definition_1: { Format.nl Format.IN }

function_definition_2: { Format.at Format.EX }
```

Defines:
function_definition, never used.
function_definition_1, never used.
function_definition_2, never used.
Uses compound_statement, n_identifier, n_lp, n_rp, optional_parameter_list, and parameter_declarations.

?? ⟨*Dummy nonterminal types* ??⟩≡ (? 0—1) ??▷

```
%type <Format.action> function_definition_1
%type <Format.action> function_definition_2
```

Uses function_definition_1 and function_definition_2.

?? ⟨optional_parameter_list ??⟩≡ (? 0—1)

```
optional_parameter_list
        : /* no formal parameters */ { Format.unitAction }
        | parameter_list { $1 }
```

Defines:
optional_parameter_list, never used.
Uses parameter_list.

?? ⟨parameter_list ??⟩≡ (? 0—1)

```
parameter_list
        : n_identifier { $1 }
        | parameter_list n_co n_identifier { $3 o $2 o $1 }
```

Defines:
parameter_list, never used.
Uses n_co and n_identifier.

?? ⟨parameter_declarations ??⟩≡ (? 0—1)

```
parameter_declarations
        : /* null */ { Format.unitAction }
        | parameter_declarations parameter_declaration { $2 o $1 }
```

Defines:
parameter_declarations, never used.
Uses parameter_declaration.

?? ⟨parameter_declaration ??⟩≡ (? 0—1)

```
parameter_declaration
        : n_int parameter_declarator_list n_sc { $3 o $2 o $1 }
```

Defines:
parameter_declaration, never used.
Uses n_int and parameter_declarator_list.

?? ⟨parameter_declarator_list ??⟩≡ (? 0—1)

```
parameter_declarator_list
        : n_identifier { $1 }
        | parameter_declarator_list n_co n_identifier { $3 o $2 o $1 }
```

Defines:
parameter_declarator_list, never used.
Uses n_co and n_identifier.

?? ⟨compound_statement ??⟩≡ (? 0—1)

```
compound_statement
        : n_lr declarations
          compound_statement_1
          statements n_rr { $5 o $4 o $3 o $2 o $1 }

compound_statement_1: { Format.nl Format.AT }
```

Defines:
compound_statement, never used.
compound_statement_1, never used.
Uses declarations, n_lr, n_rr, and statements.

?? ⟨*Dummy nonterminal types* ??⟩+≡ (? 0—1) ◁?? ??▷

```
%type <Format.action> compound_statement_1
```

Uses compound_statement_1.

?? ⟨declarations ??⟩≡ (? 0—1)

```
declarations
        : /* null */ { Format.unitAction }
        | declarations declaration { $2 o $1 }
```

Defines:
declarations, never used.
Uses declaration.

?? ⟨declaration ??⟩≡ (? 0—1)

```
declaration
        : n_int declarator_list n_sc { $3 o $2 o $1 }
```

Defines:
declaration, never used.
Uses declarator_list and n_int.

?? ⟨declarator_list ??⟩≡ (? 0—1)

```
declarator_list
        : n_identifier { $1 }
        | declarator_list n_co n_identifier { $3 o $2 o $1 }
```

Defines:
declarator_list, never used.
Uses n_co and n_identifier.

?? ⟨statements ??⟩≡ (? 0—1)

```
statements
        : /* null */ { Format.unitAction }
        | statements statement { $2 o $1 }
```

Defines:
statements, never used.
Uses statement.

?? ⟨statement ??⟩≡ (? 0—1)

```
statement
        : expression n_sc { $2 o $1 }
        | n_sc  /* null statement */ { $1 }
        | n_break n_sc { $2 o $1 }
        | n_continue n_sc { $2 o $1 }
        | n_return n_sc { $2 o $1 }
        | n_return
          statement_1
          expression n_sc { $4 o $3 o $2 o $1 }
        | compound_statement { $1 }
        | if_prefix statement { (Format.uncond Format.EX) o $2 o $1 }
        | if_prefix statement n_else statement { (Format.uncond Format.EX) o
                                                  $4 o $3 o $2 o $1}
        | loop_prefix statement { (Format.uncond Format.EX) o $2 o $1 }

statement_1: { Format.out " " }
```

Defines:
statement, never used.
statement_1, never used.
Uses compound_statement, expression, if_prefix, loop_prefix, n_break, n_continue, n_else, and n_return.

?? ⟨*Dummy nonterminal types* ??⟩+≡ (? 0—1) ◁?? ??▷

```
%type <Format.action> statement_1
```

Uses statement_1.

?? ⟨if_prefix ??⟩≡ (? 0—1)

```
if_prefix
        : n_if n_lp expression n_rp { (Format.cond Format.IN) o
                                      $4 o $3 o $2 o $1 }
```

Defines:
if_prefix, never used.
Uses expression, n_if, n_lp, and n_rp.

?? ⟨loop_prefix ??⟩≡ (? 0—1)

```
loop_prefix
        : n_while n_lp expression n_rp { (Format.cond Format.IN) o
                                         $4 o $3 o $2 o $1 }
```

Defines:
loop_prefix, never used.
Uses expression, n_lp, n_rp, and n_while.

?? ⟨expression ??⟩≡ (? 0—1)

```
expression
        : binary { $1 }
        | expression n_co binary { $3 o $2 o $1 }
```

Defines:
expression, never used.
Uses binary and n_co.

?? ⟨binary ??⟩≡ (? 0—1)

```
binary
        : n_identifier { $1 }
        | n_constant { $1 }
        | n_lp expression n_rp { $3 o $2 o $1 }
        | n_identifier n_lp optional_argument_list n_rp { $4 o $3 o $2 o $1 }
        | n_pp n_identifier %prec PP { $2 o $1 }
        | n_mm n_identifier %prec PP { $2 o $1 }
        | binary n_pl binary %prec PLUS { $3 o $2 o $1 }
        | binary n_mi binary %prec PLUS { $3 o $2 o $1 }
        | binary n_mu binary %prec TIMES { $3 o $2 o $1 }
        | binary n_di binary %prec TIMES { $3 o $2 o $1 }
        | binary n_rm binary %prec TIMES { $3 o $2 o $1 }
        | binary n_gt binary %prec GT { $3 o $2 o $1 }
        | binary n_lt binary %prec GT { $3 o $2 o $1 }
        | binary n_ge binary %prec GT { $3 o $2 o $1 }
        | binary n_le binary %prec GT { $3 o $2 o $1 }
        | binary n_eq binary %prec EQ { $3 o $2 o $1 }
        | binary n_ne binary %prec EQ { $3 o $2 o $1 }
        | binary n_an binary %prec AMP { $3 o $2 o $1 }
        | binary n_xo binary %prec CARET { $3 o $2 o $1 }
        | binary n_or binary %prec BAR { $3 o $2 o $1 }
        | n_identifier n_as binary %prec ASSIGN { $3 o $2 o $1 }
        | n_identifier n_pe binary %prec ASSIGN { $3 o $2 o $1 }
        | n_identifier n_me binary %prec ASSIGN { $3 o $2 o $1 }
        | n_identifier n_te binary %prec ASSIGN { $3 o $2 o $1 }
        | n_identifier n_de binary %prec ASSIGN { $3 o $2 o $1 }
        | n_identifier n_re binary %prec ASSIGN { $3 o $2 o $1 }
```

Defines:
binary, never used.
Uses AMP, ASSIGN, BAR, CARET, EQ, expression, GT, n_an, n_as, n_constant, n_de, n_di, n_eq, n_ge, n_gt, n_identifier, n_le, n_lp, n_lt, n_me, n_mi, n_mm, n_mu, n_ne, n_or, n_pe, n_pl, n_pp, n_re, n_rm, n_rp, n_te, n_xo, optional_argument_list, PLUS, PP, and TIMES.

?? ⟨optional_argument_list ??⟩≡ (? 0—1)

```
optional_argument_list
        : /* no actual arguments */ { Format.unitAction }
        | argument_list { $1 }
```

Defines:
optional_argument_list, never used.
Uses argument_list.

?? ⟨argument_list ??⟩≡ (? 0—1)
```
argument_list
        : binary { $1 }
        | argument_list n_co binary { $3 o $2 o $1 }
```
Defines:
argument_list, never used.
Uses binary and n_co.

3.1 Dummy nonterminals

These nonterminals are used to cause actions to be performed for tokens.

?? ⟨*Rules for dummy nonterminals* ??⟩≡ (? 0—1) ??▷
```
n_int: INT { Format.out "int\t" }
n_identifier: Identifier { Format.out $1 }
```
Defines:
n_identifier, never used.
n_int, never used.
Uses Identifier and INT.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷
```
n_lp: LPAREN { Format.out "(" }
n_rp: RPAREN { Format.out ")" }
n_co: COMMA { Format.out ", " }
```
Defines:
n_co, never used.
n_lp, never used.
n_rp, never used.
Uses COMMA, LPAREN, and RPAREN.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷
```
n_sc: SEMI { (Format.nl Format.AT) o (Format.out ";") }
```
Uses SEMI.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷
```
n_break: BREAK { Format.out "break" }
n_continue: CONTINUE { Format.out "continue" }
n_return: RETURN { Format.out "return" }
n_lr: LBRACE { (Format.at Format.IN) o
  (Format.out "{\t") o (Format.cond Format.EX) }
n_rr: RBRACE { (Format.uncond Format.AT) o
  (Format.out "}") o (Format.at Format.EX) }
```
Defines:
n_break, never used.
n_continue, never used.
n_lr, never used.
n_return, never used.
n_rr, never used.
Uses BREAK, CONTINUE, LBRACE, RBRACE, and RETURN.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷

```
n_if: IF { Format.out "if " }
n_else: ELSE { (Format.cond Format.IN) o
  (Format.out "else") o (Format.at Format.EX) }
n_while: WHILE { Format.out "while " }
n_constant: Constant { Format.out (Int.toString $1) }
```

Defines:
n_constant, never used.
n_else, never used.
n_if, never used.
n_while, never used.
Uses Constant, ELSE, IF, and WHILE.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷

```
n_pp: PP { Format.out " ++ " }
n_mm: MM { Format.out " -- " }
n_pl: PLUS { Format.out " + " }
n_mi: MINUS { Format.out " - " }
n_mu: TIMES { Format.out " * " }
n_di: DIVIDE { Format.out " / " }
n_rm: REM { Format.out " % " }
```

Defines:
n_di, never used.
n_mi, never used.
n_mm, never used.
n_mu, never used.
n_pl, never used.
n_pp, never used.
n_rm, never used.
Uses DIVIDE, MINUS, MM, PLUS, PP, REM, and TIMES.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷

```
n_gt: GT { Format.out " > " }
n_lt: LT { Format.out " < " }
n_ge: GE { Format.out " >= " }
n_le: LE { Format.out " <= " }
n_eq: EQ { Format.out " == " }
n_ne: NE { Format.out " != " }
```

Defines:
n_eq, never used.
n_ge, never used.
n_gt, never used.
n_le, never used.
n_lt, never used.
n_ne, never used.
Uses EQ, GE, GT, LE, LT, and NE.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁?? ??▷

```
n_an: AMP { Format.out " & " }
n_xo: CARET { Format.out " ^ " }
n_or: BAR { Format.out " | " }
```

Defines:
n_an, never used.
n_or, never used.
n_xo, never used.
Uses AMP, BAR, and CARET.

?? ⟨*Rules for dummy nonterminals* ??⟩+≡ (? 0—1) ◁??

```
n_as: ASSIGN { Format.out " = " }
n_pe: PE { Format.out " += " }
n_me: ME { Format.out " -= " }
n_te: TE { Format.out " *= " }
n_de: DE { Format.out " /= " }
n_re: RE { Format.out " %= " }
```

Defines:
n_as, never used.
n_de, never used.
n_me, never used.
n_pe, never used.
n_re, never used.
n_te, never used.
Uses ASSIGN, DE, ME, PE, RE, and TE.

?? ⟨*Dummy nonterminal types* ??⟩+≡ (? 0—1) ◁??

```
%type <Format.action> n_int
%type <Format.action> n_identifier
%type <Format.action> n_lp
%type <Format.action> n_rp
%type <Format.action> n_co
%type <Format.action> n_sc
%type <Format.action> n_break
%type <Format.action> n_continue
%type <Format.action> n_return
%type <Format.action> n_lr
%type <Format.action> n_rr
%type <Format.action> n_if
%type <Format.action> n_else
%type <Format.action> n_while
%type <Format.action> n_constant
%type <Format.action> n_pp
%type <Format.action> n_mm
%type <Format.action> n_pl
%type <Format.action> n_mi
%type <Format.action> n_mu
%type <Format.action> n_di
%type <Format.action> n_rm
%type <Format.action> n_gt
%type <Format.action> n_lt
%type <Format.action> n_ge
%type <Format.action> n_le
%type <Format.action> n_eq
%type <Format.action> n_ne
%type <Format.action> n_an
%type <Format.action> n_xo
%type <Format.action> n_or
%type <Format.action> n_as
%type <Format.action> n_pe
%type <Format.action> n_me
%type <Format.action> n_te
%type <Format.action> n_de
%type <Format.action> n_re
```

Uses n_an, n_as, n_break, n_co, n_constant, n_continue, n_de, n_di, n_else, n_eq, n_ge, n_gt, n_identifier, n_if, n_int, n_le, n_lp, n_lr, n_lt, n_me, n_mi, n_mm, n_mu, n_ne, n_or, n_pe, n_pl, n_pp, n_re, n_return, n_rm, n_rp, n_rr, n_te, n_while, and n_xo.

4 Trailer

Empty trailer.

?? ⟨ *Trailer* ?? ⟩≡ (? 0—1)

5 Limitations

All the token types should really be augmented with position information passed by the lexer, in order to be able to generate informative error messages.

6 Indices

6.1 Chunks

6.2 Identifiers

References

[1] Axel T. Schreiner and H. George Friedman, Jr. *Introduction to Compiler Construction with UNIX*[1] . Prentice-Hall, Inc., New Jersey, 1985.

[1] UNIX is a trademark of Bell Laboratories.