Compiler project tasks — part 2, due Friday 21 September

Below you will find a semiformal description of the syntactic structure of our source language, as context-free grammar. You are to write a parser. The parser will be invoked from the command line as follows: `parse <filename>`; for example, `parse primes.m`. The parser should invoke the scanner that you wrote in the preceding project phase, and should work with the tokens returned by the scanner.

The output should be written to a listing file, for example, `primes.parse`. The listing should represent the parse tree of the input program (assuming it is correct), turned sideways: each line contains one non-terminal or terminal symbol, and the indentation of the line corresponds to the depth of the symbol in the parse tree. For terminal symbols which carry meaning (such as identifiers and numbers), the name of the symbol should be followed by the literal text of the symbol. All symbols should be followed by an indication of their coordinates. For example:

```
identifier  Sqrt  (25,30)-(25,33)
```

means that at line 25 of the source file at columns 30 through 33 there is a token `Sqrt`, which is an identifier.

In addition to this printed output, you should construct the parse tee internally, in a form suitable for further analysis. Details will depend on your choice of implementation language and strategy.

Syntactic structure of the source language

Compilation Unit Productions

Program ::= Module

Module ::= **`module`** *identifier semi* BlockNoScope *identifier dot*

Block ::= BlockNoScope

BlockNoScope ::= OptDecls **`begin`** Stmts **`end`**

OptDecls ::= ε | Decls

Decls ::= Declaration | Decls Declaration

Declaration ::= **`const`** OptConstDecls | **`type`** OptTypeDecls | **`typerec`** RecTypeDecls | **`var`** OptVarDecls | ProcDecl

RecTypeDecls ::= TypeDecl | RecTypeDecls **`and`** TypeDecl

OptConstDecls ::= ε | ConstDecls

OptTypeDecls ::= ε | TypeDecls

OptVarDecls ::= ε | VarDecls

ConstDecls ::= ConstDecl | ConstDecls ConstDecl

TypeDecls ::= TypeDecl | TypeDecls TypeDecl

VarDecls ::= VariableDecl | VarDecls VariableDecl

ConstDecl ::= *identifier colon* Type *equal* Expr *semi* | *identifier equal* Expr *semi*

TypeDecl ::= *identifier equal* Type *semi*

VariableDecl ::= IDList *colon* Type *semi* | IDList *colon* Type *coloneq* Expr *semi* | IDList *coloneq* Expr *semi*

ProcDecl ::= **`procedure`** *identifier* Signature *equal* BlockNoScope *identifier semi*

Signature ::= *lpar* OptFormals *rpar* OptDeclType

OptDeclType ::= ε | *colon* Type

OptFormals ::= ε | Formals | Formals *semi*

Formals ::= Formal | Formals *semi* Formal

Formal ::= OptMode IDList *colon* Type

OptMode ::= ε | **`value`** | **`var`**

Statement Productions

Stmts ::= ε | StmtList | StmtList *semi*

StmtList ::= Stmt | StmtList *semi* Stmt

Stmt ::= Block | AssignStmt | CallStmt | ExitStmt | EvalStmt | ForStmt | IfStmt | LoopStmt | ReadStmt | RepeatStmt | ReturnStmt | WhileStmt | WriteStmt

AssignStmt ::= Expr *coloneq* Expr

CallStmt ::= ProcCall

ExitStmt ::= **`exit`**

EvalStmt ::= **`eval`** Expr

ForStmt ::= **`for`** *identifier coloneq* Expr **`to`** Expr OptBy **`do`** Stmts **`end`**

OptBy ::= ε | **`by`** Expr

IfStmt ::= IfHead **`end`** | IfHead **`else`** Stmts **`end`**

IfHead ::= **`if`** Expr **`then`** Stmts | IfHead **`elsif`** Expr **`then`** Stmts

LoopStmt ::= **`loop`** Stmts **`end`**

ReadStmt ::= **`read`** *lpar text comma* ReadList *rpar*

ReadList ::= Expr | Expr *comma* ReadList

RepeatStmt ::= **`repeat`** Stmts **`until`** Expr

ReturnStmt ::= **`return`** | **`return`** Expr

WhileStmt ::= **`while`** Expr **`do`** Stmts **`end`**

WriteStmt ::= **`write`** *lpar text* WriteList *rpar*

WriteList ::= ε | *comma* Expr WriteList

Type Productions

Type ::= ArrayType | RecordType | SubrangeType | *identifier* | *lpar* Type *rpar* | **`ref`** Type

ArrayType ::= **`array`** OptTypeList **`of`** Type

OptTypeList ::= ε | TypeList

TypeList ::= Type | TypeList *comma* Type

RecordType ::= **`record`** OptFields **`end`**

OptFields ::= ε | Fields | Fields *semi*

Fields ::= IDList *colon* Type | Fields *semi* IDList *colon* Type

SubrangeType ::= *lbrack* Expr *dotdot* Expr *rbrack*

Expression Productions

Expr ::= Expr **`or`** Expr | Expr **`and`** Expr | **`not`** Expr | Expr Relop Expr | Expr Addop Expr | Expr Mulop Expr | Sign Expr | Primary

Relop ::= *equal* | *nequal* | *less* | *greater* | *le* | *ge*

Addop ::= *plus* | *minus*

Mulop ::= *star* | **`div`** | **`mod`**

Sign ::= *plus* | *minus*

Primary ::= *identifier* | *integer* | *real* | *text* | *lpar* Expr *rpar* | Primary *dot identifier* | ArrayRef | ProcCall | Primary *caret* | **`new`** *lpar* Type *rpar* | **`nil`** *lpar* Type *rpar*

ArrayRef ::= Primary *lbrack rbrack* | ArrayHead *rbrack*

ArrayHead ::= Primary *lbrack* Expr | ArrayHead *comma* Expr

ProcCall ::= Primary *lpar rpar* | CallHead *rpar*

CallHead ::= Primary *lpar* Expr | CallHead *comma* Expr

Miscellaneous Productions

IDList ::= *identifier* | IDList *comma identifier*