

CS 415 Project - Spring 2000

Department of Computer Science
Rutgers University

Due 7pm, Wednesday 3/1/2000

1 Building a Parser

Using bison, you are to write a parser for our language. As a reminder, our language is a subset of Pascal. This subset differs from Pascal in the following ways. Program headings contain no list of input and output files. Blocks have only variable and procedure/function declarations and compound statements. Types are limited to the scalars: integer, character, boolean; and single dimensional arrays of scalars indexed by integers. Only the following statements are included: while, if, procedure call, assignment, and compound. Operators are restricted to arithmetic, logical, and relational.

Here's the grammar for our language:

```
start      ::= program ID ; block .
block      ::= variables procdcls cmpdstmt
procdcls   ::= procdcls procdcl | <empty string>
procdcl    ::= procedure ID parmlist ; block ;
           | function ID parmlist : stype ; block ;
parmlist   ::= ( parms ) | <empty string>
parms      ::= parms ; parm | parm
parm       ::= var vardcl | vardcl
variables  ::= var vardcls | <empty string>
vardcls    ::= vardcls vardcl ; | vardcl ;
vardcl     ::= idlist : type
idlist     ::= idlist , ID | ID
type       ::= array [ integer_constant .. integer_constant ] of stype | stype
stype      ::= integer | char | boolean
stmtlist   ::= stmtlist ; stmt | stmt
stmt       ::= ifstmt | wstmt | astmt | procstmt | cmpdstmt | writestmt
writestmt  ::= writeln ( exp )
procstmt   ::= ID optexplist
ifstmt     ::= if exp then stmt else stmt | if exp then stmt
wstmt      ::= while exp do stmt
cmpdstmt   ::= begin stmtlist end
astmt      ::= lvalue := exp
optexplist ::= ( explist ) | <empty string>
explist    ::= explist , exp | exp
```

```

exp      ::= exp + exp | exp - exp | - exp | exp * exp | exp div exp
          | exp != exp | exp == exp | exp >= exp | exp > exp | exp < exp
          | exp <= exp | exp and exp | exp or exp | exp exor exp
          | not exp | ( exp ) | ID ( explist ) | lvalue | constant
lvalue   ::= ID | ID [ exp ]
constant ::= integer_constant | char_constant | true | false

```

Your project should produce a listing, identify any syntax errors, and print a list of identifiers (not including reserved words) and operators encountered during the parse. Please see the demo files and their output for the format of this interface. You must detect at least the following syntax errors: illegal procedure declaration, illegal parameter, illegal variable declaration, illegal statement, and illegal expression. You must enforce the following precedence and associativity for each operator:

Operator	Precedence	Associativity
Relational operators	lowest	not associative
+ - or		left associative
* div and		left associative
not unary-	highest	right associative

2 What To Do

To get started, you should download `project2.tar.gz` from the class web. This will provide you with:

- A Makefile,
- A skeleton `parse.y` file that you should modify,
- A simple symbol table that you can use to keep track of identifiers (`syntab.h` and `syntab.c`), and
- Several demo files containing sample programs.

You must use the scanner that you built for phase 1 of the project. Please use `TRUE_TOK` and `FALSE_TOK` instead of `TRUE` and `FALSE` since the latter are predefined in the C compiler.

Similar to the last phase, you compile and link your parser by typing `make`. For this project you should only need to change the file `parse.y`. All questions regarding the project should be posted in our news group `ru.nb.dcs.class.415`.

Remember, a good source of information are the man pages (e.g., `man bison`) and the recommended book on `lex` and `yacc`.