

Heriot-Watt University
School of Mathematical and Computer Science
Compilers (F23PF2) Assignment 1

Murdoch Gabbay

1 Introduction

Please build a syntax analyzer for a simple imperative programming language.

This exercise is intended to give you experience of the UNIX tools LEX¹ and YACC.

2 Language Specification

The grammar of your imperative programming language is defined in BNF notation as follows:

```
 $\langle program \rangle ::= \text{PROG } \langle command \rangle \text{ ENDPROG}$   
 $\langle command \rangle ::= \langle variable \rangle := \langle term \rangle \mid$   
          BEGIN  $\langle declarations \rangle$   $\langle commands \rangle$  END  $\mid$   
          IF  $\langle statement \rangle$  THEN  $\langle command \rangle \mid$   
          IF  $\langle statement \rangle$  THEN  $\langle command \rangle$  ELSE  $\langle command \rangle \mid$   
          WHILE  $\langle statement \rangle$  LOOP  $\langle command \rangle \mid$   
          READ(  $\langle variable \rangle$  )  $\mid$   
          PRINT(  $\langle term \rangle$  )  
 $\langle declarations \rangle ::= \text{VAR } \langle variable \rangle ; \langle declarations \rangle \mid \epsilon$   
 $\langle commands \rangle ::= \langle command \rangle ; \langle commands \rangle \mid \langle command \rangle$ 
```

Note that:

- A program *variable* is denoted by a letter followed by a possibly empty sequence of letters and digits, *e.g.*
 x, sum, Radius1
- A *term* denotes a value; in our simple programming language all values are numbers. Compound terms are built out of variables and constants using the standard arithmetic operators, *e.g.*
 (y * z) + 2
- A *statement* has a truth-value which is either *true* or *false*. Some example atomic statements follow:
 TRUE, FALSE, x = 1, x < y
Compound statements are built out of atomic statements using the standard logical operators, *e.g.*
 (x = y AND y = z) OR NOT(x = 0)
- In a block command of the form BEGIN VAR D1; ... ; VAR Dn; C END, the scope of the declarations D1 to Dn extends over the command C, *i.e.* the scope is delimited by the BEGIN ... END. Note that the language supports nested blocks and the *nested block structure* scoping rules apply – see “Symbol Table Management I” lecture notes.

3 Detailed work plan

This assignment requires you to achieve the following tasks:

1. Construct a lexical analyzer for the specified language using LEX;
2. Building upon your lexical analyzer, construct a syntax analyzer for the specified language using YACC. Your syntax analyzer should report success or failure. In the case of failure you should attempt to provide ‘useful’ diagnostic information to the user.

¹You can use either LEX or FLEX.

3. Extend your syntax analyzer with a symbol table mechanism. You decide how sophisticated it should be (see Section 5).
4. Test your syntax analyzer on correct and incorrect programs, and apply your syntax analyzer to the test data supplied online at www.gabbay.org.uk.

4 Deliverables

Your submission should include:

- a description of how you resolved any language issues such as ambiguity.
- your BNF definition for a *term* and a *statement*, *i.e.* as implemented by your syntax analyzer.
- the LEX source code for your lexical analyzer;
- the YACC source code for your syntax analyzer which exploits your LEX generated lexical analyzer;
- a description of your symbol table mechanism (optional, see Section 5).
- a manual page describing how to build and operate your syntax analyzer;
- documentation of your testing, in particular with regard to the test data supplied at lecture. Clearly indicate which programs are classified by your syntax analyzer as correct and incorrect.
- **Big fat warning:** When you write up, keep your english short but conversational, and gramatically correct. I *will* mark down a project if the prose is so poorly-edited that it is hard to read.

5 Marking Scheme

The “symbol table mechanism” is optional. You can get a B-grade without it, but to get an A-grade you must do that part too.

6 Submission Procedure

Please submit this assignment via the course work box in the crush area by 10:30am on Wednesday 30th of May 2007, and send me a pdf by e-mail.

Another big fat warning: this is an individual project. Your submission must be your own work. If it is not, I may tear up your script or worse.

This assignment counts for 10% of the final assessment of this module.

7 Some Hints

- The choice of end-of-input marker is yours, though end-of-file is the usual option. Note that `yylex`, *i.e.* a LEX-generated lexer, returns the value 0 when it reaches the end-of-file.
- The simplest way of allowing your syntax analyzer to read data from a file is via the UNIX file redirection command, *e.g.*

```
my-syntax-analyzer.o < my-test-data-file
```
- Note that `BEGIN` is a LEX reserved word. Defining `BEGIN` as a token within LEX can cause problems. One solution is to prefix all tokens with the letter T, *e.g.* `T_BEGIN`, `T_END`, `T_WHILE`, . . . , but if you find that tiresome think up your own scheme.
- Build your syntax analyzer incrementally, *i.e.* begin with simple definitions for *term* and *statement*.
- Though this is an individual project and the work must be your own, you can discuss technical points with each other online if you wish — and you’ll be helping each other if you do.