Prolog - lab

What to think about

The Parser: start code

- **Parser: read source code + lexical analysis + syntax analysis**
- Prolog code: **Reader + Lexer + Parser**

```
ParseFile(File, Result) :-
    read_in(File, L), lexer(L, Tokens), parser(Tokens, Result).
```

file ➔ lexemes ➔ tokens

- This is turn may be packaged using tail recursion

```
testa :- parseFiles(['testok1.pas', 'testok2.pas', 'testok3.pas']).
parseFiles([]).
parseFiles([H|T]) :- write('Testing '),  write(H), nl,
    read_in(H,L),  lexer(L, Tokens), parser(Tokens, _), nl,
    write(H), write(' end'), nl, nl, parseFiles(T).
```

Prolog parser: reader

```
                                    // instantiations
read_in(File,[W|Ws]) :-             // File          in parameter
        see(File),                   // File          in parameter
        get0(C),                    // C             out parameter
        readword(C, W, C1),         // W, C1         out parameters
        restsent(W, C1, Ws),        // Ws            out parameter
        nl,
        seen.
```

NB: which LHS parameters are instantiated and when?

Prolog parser: readword

```
readword(C, W, _)      :- C = -1, W = C.          /* EOF */
readword(C, W, C2)     :- C = 58, …               /* ":" or ":=" */

readword(C, W, C1)     :- single_character(C),
                          name(W, [C]), get0(C1).

readword(C, W, C2)     :- in_word(C, NewC),       /* alpha & num */
                          get0(C1), restword(C1, Cs, C2),
                          name(W, [NewC|Cs]).

readword(_, W, C2)     :- get0(C1), readword(C1, W, C2).
```

What you have to do #1

```
readword(C, W, C2) :- C = 58, get0(C1), readwordaux(C, W, C1, C2).

readwordaux(C, W, C1, C2) :- C1   = 61, name(W, [C, C1]), get0(C2).
readwordaux(C, W, C1, C2) :- C1 \= 61, name(W, [C]), C1 = C2.
```

Implement the above

Explain what this does and how it works – in detail (5p)

What are C, W, C1, C2?

What does C1 = C2 mean? A slightly tricky question!

What does name do?

In which order are the variables instantiated?

What you have to do #2

```
readword(C, W, C2)     :- in_word(C, NewC),    /* alpha & num */
                          get0(C1),  restword(C1, Cs, C2),
                          name(W, [NewC|Cs]).
```

This would be better as 2 rules

(1) Find an alphanumeric lexeme

(2) Find a number lexeme

Decide the best order for these!

What you have to do #3

note the use of **273** (for undefined symbols) and **275** (EOF)

The Lexer returns these **token values**.

When and how are these generated?

Hint: look at the output file for the parse.

What does the reader have to do in each case?

Hint: it is not whitespace!

Prolog parser: restword

```
restword(C, [NewC|Cs], C2) :-
        in_word(C, NewC),
        get0(C1),
        restword(C1, Cs, C2).

restword(C, [], C).
```

Q1: what is the fail condition?

Q2: what is the stop condition?

Prolog parser: reader

(1) **Test for EOF** - handle EOF

(2) **Test for 58 (':')** - handle ':' and ':='

(3) **Test for a single character** - handle single character

(4) **Test for a word** - handle a word – NB change!

(5) ➔ **all other characters (e.g. space, tab, CR, LF) – IGNORE**

(5) Is the default condition (usually recursive)

In Prolog predicates list and identify the various cases

Prolog parser: restsent

```
restsent(W, _, [])          :- W = -1.                  /* EOF */
restsent(W, _, [])          :- lastword(W).             /*  '.'  */
restsent(_, C, [W1|Ws]) :-
        readword(C, W1, C1),
        restsent(W1, C1, Ws).       /* rest of sentence */
```

What are the stop condition(s)?

Which stop condition is always guaranteed?

What are the fail condition(s)?

Why was the predicated designed this way?

The simplified abstract view

- **The program text**

```
program testok1(input, output);
    var a, b, c: integer;
    begin
      a := b + c * 2
      end.
```

- **Program = list of lexemes**
  - **Head = program**
  - **Tail = [testok1, (, input, ,, output,), ;, var, a, ,, b, ,, c, :, integer, ;, begin, a, :=, b, +, c, *, 2, end, .]**
- **Lexeme = list of characters** e.g. [p, r, o, g, r, a, m]
  - **Head = p**
  - **Tail = [r, o, g, r, a, m]**

Prolog parser: lexer

- **Converts lexemes to tokens**
- **Handles the 273 and 275 cases!**

- **Think: what are the legal terminal symbols handled by the lexer?**

- **How will alphanumeric and numeric strings be handled here?**

- **Which auxiliary predicates do you need to define?**

Prolog parser: parser

- **Terminal symbols**

  program → [256].

  …

  scolon → [59].

  assign_op → [271].

  etc.

- **Non-Terminal symbols**

  program → header, var_part, stat_part.

  etc.

Lab 2 – thinking part!

- #1

  ‘:’ and ‘:=‘

- #2

  Alphanumeric and number lexemes

- #3

  How to handle 273 and 275?