Lab Instructions for
Programming Language Translation

Introduction

In this lab assignment, studens will be confronted with many of the central problems of lexical and syntax analysis of programming languages on a simple example programming language we'll call *PLT*.

PLT is defined informally through the example program shown in Listing 1.

```
function isPrime(n) {
  if (n == 2) return true;
  else if (n<2 || n%2==0) return false;
  for (d=3; d*d <= n; d=d+2) {
    if (n%d == 0) {
      return false;
    }
  }
}

function main() {
  n = input_int();
  for (i=0; i<=n-1; ++i) {
    if (isPrime(i)) {
      output_int(i);
    }
  }
  return 0;
}
```

Listing1: Example program in *PLT* that prints out primes up to some upper limit.

PLT's syntax is in the C-family. The only supported control statements are `if/else` and a `for` loop that have the same form as in C and other C-family languages. Furthermore, only simple assignment via the = operator is supported.

The only type in *PLT* is `int`, and there are no variable declarations. This is omitted as it doesn't impact lexical and syntactic analysis in any significant way. Accordingly, only decimal integer constants are supported.

Expressions are similar to those in other languages, except the operator set is restricted to binary addition (+), subtraction (-), multiplication (*), division (/) and remainder (%) with unary minus (-) and pre/post increment/decrement operators (++ and --). The operators have the same priority as in other languages, and a different order of computation can be achieved by using braces.

For logical tests, the value zero evaluates to false and all other values evaluate to true. Both binary and (&&) and binary or (||) are supported.

The keywords in the language are `function`, `if`, `else`, `for`, `return`, `true` and `false`.

Assignment 1: Lexical Analysis

Your assignment is to build a lexical analyzer for *PLT* using a lexer-generator like Lex. You can use any lexer-generator you like. Different lexer-generators output lexers in different programming languages and you will likely need to write some code fragments in this language, so make that the primary criterion when choosing a lexer-generator. Additionally, as you will be using a parser-generator in the second assignment, choose a lexer-generator that is compatible with some parser-generator so that you don't have to redefine your lexer later.

The basic steps in completing this assignment are as follows:

1. define the appropriate tokens for *PLT*
2. write the lexer rules that will tokenize the input *PLT* code
3. use the lexer-generator to create the lexer itself

Given a *PLT* source file as input, the lexer should output the token stream of the input source along with the identified lexemes. For error recovery, use the "panic mode" strategy described in the textbook.

Your work will be evaluated by hand on several example programs, and you will be expected to be able to answer questions about the areas of lexical analysis that were covered in the assignment and also to explain the lexer rules you wrote.

Assignment 2: Syntax Analysis

The second assignment is to build a syntax analyzer for *PLT* using a parser-generator like Yacc. When choosing a parser-generator, similar considerations apply as in the first assignment.

The basic steps in completing this assignment are as follows:

1. define the syntax of *PLT* using the rules of the parser-generator (for most parser-generators, these rules closely resemble either an LL(1) or an LR(1) grammar with a few possible ambiguities that are solved using special rules/actions that are somewhat specific to the particular parser-generator but are based on the same principles that are covered in the textbook and those used in Yacc)

2. use the parser-generator to generate a parser (i.e. a syntax analyzer) for *PLT*

The syntax analyzer should output a parse tree of the token stream that it is given as input. In most cases, parser-generators also include facilities for lexical analysis and you will most likely be able to reuse the rules you've written in the first assignment to perform lexical analysis. In that case, the input to the program would be a source file and the output would be a parse tree. Both options are acceptable and equally valid.

Similarly to the first assignment, evaluation will be done by hand by a TA and you will he quizzed on topics of syntax analysis that you've encountered during the assignment and on the parsing rules you've devised.