TDDD55- Compilers and Interpreters
Lesson 1

Zeinab Ganjei
(zeinab.ganjei@liu.se)

Department of Computer and Information Science
Linköping University

Purpose of Lessons

- Practice theory
- Introduce the laboratory assignments
- Prepare for the final examination.

Prepare by reading the laboratory instructions, the course book, and the lecture notes.

All the laboratory instructions and material available in the ***course directory, ~TDDD55/lab/***

or on the course homepage.

Laboratory Assignments

- In the laboratory exercises you should get some practical experience in compiler construction.
- There are 4 separate assignments to complete in 4x2 laboratory hours. You will also (most likely) have to work during non-scheduled time.

Lessons Schedule

- Formal languages and automata theory
- Formal languages and automata theory, Flex
- Intermediate code generation, Bison
- Exam preparation

Handing in and deadline

- Demonstrate the working solutions during scheduled sessions.
- Then, hand in code and answers to any questions via e-mail. One e-mail from your LiU-email per group (subject: *TDDD55: lab no.*).

- Deadline for all the assignments is: **December 14 2017**.

- Sign up in the webreg!

Laboratory Assignments

- Lab 1 Attribute Grammars and Top-Down Parsing
- Lab 2 Scanner Specification
- Lab 3 Parser Generators
- Lab 4 Intermediate Code Generation

1. Attribute Grammars and Top-Down Parsing

- Some grammar rules are given
- Your task:
  - Rewrite the grammar (eliminate left recursion, etc.)
  - Add attributes and attribute rules to the grammar
  - Implement your attribute grammar in a C++ class named **Parser**. The **Parser** class should contain a method named **Parse** that returns the value of a single statement in the language.

2. Scanner Specification

- Finish a scanner specification given in Flex(scanner.l), by adding rules for comments, identifiers, integers, and reals.
- More details in lesson 2.

3. Parser Generators

- Finish a parser specification given in Bison (parser.y), by adding rules for expressions, conditions and function definitions, ... You also need to augment the grammar with error productions.
- More details in lesson 3.

4. Intermediate Code Generation

- The purpose of this assignment to learn about how parse trees can be translated into intermediate code.
- Finish a generator for intermediate code by adding rules for some language statements.
- More details in lesson 3

Hints for Laboratory Assignment 1

Grammar for simple mathematical expressions

S -> E <end of line> S	Single expression
| <end of file>	No more input
E -> E + E	Addition
| E - E	Subtraction
| E * E	Multiplication
| E / E	Division
| E ^ E	Exponentiation
| - E	Unary minus
| (E)	Grouping
| id (E)	Function call

Not Suitable for a Top-Down Technique

- No operator precedence
  - e.g E -> E + E
    | E * E
- No operator associativity
  - e.g. E ^ E
- Left recursion
  - e.g. E -> E + E
- Ambiguity

Rewriting the Grammar

- Use one non-terminal for each precedence level.

  E ::= E + E | E – E | T

  T ::= T * T | T / T

- (Left) Associativity: using (left-)recursive production

  E ::= E + E | E – E | T => E ::= E + T | E – T | T

See for instance:
http://www.lix.polytechnique.fr/~catuscia/teaching/cg428/02Spring/lecture_notes/L03.html

Rewriting the Grammar (2)

- The grammar obtained so far has left recursion
  - Not suitable for a predictive top-down parser
- Transform the grammar to right recursive form:

  A ::= A α | β (where β may not be preceded by A)

  is rewritten to

  A ::= β A'

  A' ::= α A' | ε

See *Lecture 5 Syntax Analysis, Parsing*

More details: http://en.wikipedia.org/wiki/Left_recursion

Attribute Grammars

- Define attributes for the productions of a formal grammar
- Example:

```
S ::= E     { display( E.val ); }
E ::= E + T   { E.val = E.val + T.val; }
      | T           { E.val = T.val;  }
T ::= T * F   { T.val = T.val * F.val; }
      | F           { T.val = F.val; }
F ::= ( E )     { F.val = E.val; }
      | num { F.val = num.val; }
```

Implementation: main.cc

```
int main(void) {
        Parser parser;
        double val;
        while (1) {

                try   {
                  cout << "Expression: " << flush;
                  val = parser.Parse();
                          cout << "Result:     " << val << '\n' << flush;
                }
                 catch (ScannerError& e) {
                        cerr << e << '\n' << flush;
                        parser.Recover();
                }
                        catch (ParserError) { parser.Recover(); }

                         catch (ParserEndOfFile)  {
                        cerr << "End of file\n" << flush; exit(0); }
                }
        }
}
```

Implementation: lex.cc and lex.hh

- The files lex.cc and lex.hh implement the lexer
- You don’t need to change anything in those files.

Implementation : lab1.cc, lab1.hh

```
double Parser::Parse(void) {
        Trace x("Parse");
        double val = 0;
        current_token = scanner.Scan();
        switch (current_token.type)
        {
                case kIdentifier:
                case kNumber:
                case kLeftParen:
                case kMinus:
                        val = pExpression();
                        if (current_token.type != kEndOfLine)
                                throw ParserError();
                default:
                        throw ParserError();
        }
        return val;
}
```

Implementation…

- Add one function for each non-terminal in the grammar to your *Parser* class.
- Also implement some simple error recovery in your *Parser* class.
- See Lecture 5 for details.

```
double Parser::pExpression(void) {
    switch (current_token.type) {
    ...
    }
}
```

Laboratory skeleton

`~TDDD55`

`/lab`

`/doc`

Documentation for the assignments.

`/lab1`

Contains all the necessary files to complete the first assignment

`/lab2`

Contains all the necessary files to complete the second assignment

`/lab3-4`

Contains all the necessary files to complete assignment three and four

Installation

- Take the following steps in order to install the lab skeleton on your system:
  - Copy the source files from the course directory onto your local account:

```
mkdir TDDD55
cp -r ~TDDD55/lab TDDD55
```

  - You might also have to load some modules (more information in the laboratory instructions).