Formal Languages and Compilers

Lab 12

FLEX & Bison in language libraries
Bison in Java(JavaCC)

- Java Compiler Compiler (JavaCC) is a parser generator which also includes tree building (via a tool called JJTree included with JavaCC), actions and debugging.
- Allows EBNF specifications
- The lexical and grammar specifications are written in the same file
- The lexical analyzer of JavaCC can handle full Unicode input
- JJDoc: a tool that converts grammar files to documentation files, optionally in HTML.
- https://javacc.github.io/javacc/documentation/

FLEX & Bison in language libraries
Python LEX-YACC(PLY)

PLY is an implementation of lex and yacc parsing tools for Python

In a nutshell, PLY is nothing more than a straightforward lex/yacc implementation. Here is a list of its essential features:

- It's implemented entirely in Python.
- It uses LR-parsing which is reasonably efficient and well suited for larger grammars.
- PLY provides most of the standard lex/yacc features including support for empty productions, precedence rules, error recovery, and support for ambiguous grammars.
- PLY is straightforward to use and provides *very* extensive error checking.
- PLY doesn't try to do anything more or less than provide the basic lex/yacc functionality. In other words, it's not a large parsing framework or a component of some larger system.
- How to use it:
  - pip install ply (might not be supported anymore)
  - Download the package from here

FLEX & Bison in language libraries
Python LEX-YACC(SLY)

- SLY is library for writing parsers and compilers
- SLY requires Python 3.6 or newer
- Loosely based on the traditional compiler construction tools lex and yacc and implements the same LALR(Look-Ahead Left-to-Right parser) parsing algorithm
- Most of the features available in lex and yacc are also available in SLY
- It provides two separate classes Lexer and Parser:
  - The Lexer class is used to break input text into a collection of tokens specified by a collection of regular expression rules
  - The Parser class is used to recognize language syntax that has been specified in the form of a context free grammar
  - The two classes are typically used together to make a parser

Example SLY

```python
from sly import Lexer, Parser

class CalcLexer(Lexer):
    tokens = { NAME, NUMBER, PLUS, MINUS, ASSIGN}
    ignore = ' \t'

    # Tokens
    NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
    NUMBER = r'\d+'

    # Special symbols
    PLUS = r'\+'
    MINUS = r'-'
    ASSIGN = r'='
```

```python
class CalcParser(Parser):
    tokens = CalcLexer.tokens

    def __init__(self):
        self.names = { }

    @_('NAME ASSIGN expr')
    def statement(self, p):
        self.names[p.NAME] = p.expr

    @_('expr')
    def statement(self, p):
        print(p.expr)

    @_('expr PLUS expr')
    def expr(self, p):
        return p.expr0 + p.expr1

    @_('expr MINUS expr')
    def expr(self, p):
        return p.expr0 - p.expr1

    @_('NUMBER')
    def expr(self, p):
        return int(p.NUMBER)
```

```python
if __name__ == '__main__':
    lexer = CalcLexer()
    parser = CalcParser()
    while True:
        try:
            text = input('calc > ')
        except EOFError:
            break
        if text:
            parser.parse(lexer.tokenize(text))
```

```
PS D:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex\Ex8_l4>  d:; cd 'd:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex\Ex8_l4'; & 'C:\Users\ciung\anaconda3\python.exe
' 'c:\Users\ciung\.vscode\extensions\ms-python.python-2021.12.1559732655\pythonFiles\lib\python\debugpy\launcher' '54069' '--' 'd:\Facultate\Predat\FormalLanguagesAndCompilers\L
ab\Ex\Ex8_l4\calc.py'
WARNING: 4 shift/reduce conflicts
calc > 4-5
-1
calc > 6+7
13
calc >
```

Exercise

- Add multiplication and division
- Add mathematical parenthesis

Homework

- Add power (n^m)
- Add factorial (n!)