Processamento de Linguagens
Engenharia Informática (3º ano)
Trabalho Prático 1

29 de Março de 2022 (9h00)

Dispõe de **7 semanas** para desenvolver este trabalho, a entrega deverá ser feita até à meia noite de **15 de Maio**.

1 Objectivos e Organização

Este trabalho prático tem como principais objectivos:

- aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical), reforçando a capacidade de escrever gramáticas, quer independentes de contexto (GIC), quer tradutoras (GT);
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, a partir de uma gramática tradutora;
- desenvolver um compilador gerando código para um objetivo específico;
- utilizar geradores de compiladores baseados em gramáticas tradutoras, concretamente o Yacc, versão PLY do Python, completado pelo gerador de analisadores léxicos Lex, também versão PLY do Python.

Na resolução dos trabalhos práticos desta UC, aprecia-se a imaginação/criatividade dos grupos em todo o processo de desenvolvimento!

Deve entregar a sua solução até Domingo dia 15 de Maio. O ficheiro com o relatório e a solução deve ter o nome 'pl2022-tp2-grNN', em que NN corresponderá ao número de grupo. O número de grupo já foi atribuído no registo do TP1.

Cada grupo, **é livre** para escolher qual o enunciado que pretende desenvolver.

A submissão deverá ser feita por email com os seguintes dados:

to: jcr@di.uminho.pt

subject: PL2022::grNN::TP1::Enunciado

body: Colocar um ZIP com os ficheiros do TP1: relatório, código desenvolvido e datasets de teste.

Em cima, "Enunciado"é um dos valores: Pandoc, Ply-Simple, RecDesc, TDTabela.

Na defesa, a realizar na semana de 16 a 20 de Maio, o programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, em data e hora a marcar. O relatório a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da solução e sua implementação, deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX.

2 Enunciados

2.1 Linguagem de templates (inspirada nos templates Pandoc)

Considere a linguagem de definição de templates do Pandoc:
(ver exemplos e detalhes em `https://pandoc.org/MANUAL.html#templates`)

- Escrever a gramática (comece por um subconjunto e vá sucessivamento enriquecendo).
- Construir o analisador léxico.
- Construir o parser que:
- hipótese 1: dado um template T1 gerar uma função python `expand_T1` que recebendo um dicionário dê um "texto final"
- hipótese 2: escreve um programa que dado um template T1 e um dicionário produza o "texto final"
- hipótese 3: escreve um programa que dado um ficheiro template (T1) e um ficheiro YAML (dicionário) produza o "texto final"

2.2 Tradutor PLY-simple para PLY

O gerador de parsers PLY, embora poderoso tem uma sintaxe um pouco complexa. Pretendemos criar um tradutor que a partir de uma sintaxe mais "limpa"PLY-simple, gere as funções PLY convenientes.

- Considere o exemplo abaixo. Escreva manualmente um versão análoga em PLY (o mais paralela que for possível); corrija e adapta onde for necessário.
- Deduza um esquema de tradução.
- Escreva um compilador que traduza PLY-simple em PLY!

Exemplo de uma sintaxe possível (pode alterar a sintaxe):

```
%% LEX
%literals = "+-/*=()"          ## a single char
%ignore   = " \t\n"
%tokens   = [ 'VAR','NUMBER' ]

[a-zA-Z_][a-zA-Z0-9_]*  return('VAR', t.value)
\d+(\.\d+)?             return('NUMBER', float(t.value)
.                       error(f"Illegal character '{t.value[0]}', [{t.lexer.lineno}]",
                              t.lexer.skip(1) )

%% YACC

%precedence = [
    ('left','+','-'),
    ('left','*','/'),
    ('right','UMINUS'),
  ]

# symboltable = dictionary of variables
ts = { }

stat : VAR '=' exp           { ts[t[1]] = t[3] }
stat : exp                   { print(t[1]) }
exp  : exp '+' exp           { t[0] = t[1] + t[3] }
exp  : exp '-' exp           { t[0] = t[1] - t[3] }
exp  : exp '*' exp           { t[0] = t[1] * t[3] }
exp  : exp '/' exp           { t[0] = t[1] / t[3] }
exp  : '-' exp %prec UMINUS  { t[0] = -t[2] }
exp  : '(' exp ')'           { t[0] = t[2] }
exp  : NUMBER                { t[0] = t[1] }
exp  : VAR                   { t[0] = getval(t[1]) }

%%

def p_error(t):
    print(f"Syntax error at '{t.value}', [{t.lexer.lineno}]")

def getval(n):
    if n not in ts: print(f"Undefined name '{n}'")
    return ts.get(n,0)

y=yacc()
y.parse("3+4*7")
```

2.3 Gerador de Parsers LL(1) Recursivos Descendentes

Apesar de algumas limitações das gramáticas LL(1) um parser recursivo descendente tem a vantagem de poder ser implementado em qualquer linguagem de programação que suporte recursividade.

A tarefa de escrever um parser recursivo descendente é bastante repetitiva e segue um padrão algorítmico, padrão esse que se quer captar, nesta proposta, na geração de código.
Assim, neste projeto deverás realizar as seguintes tarefas:

- Criar uma linguagem para descrição de gramáticas independentes de contexto;
- Criar um reconhecedor para essa linguagem que:
  1. Verifique se a linguagem é LL(1);
  2. Caso seja, gere o código Python que implementa o parser recursivo descendente;
  3. Como extra, poderás pensar em arranjar uma maneira de juntar ações semânticas ao parser gerado.

2.4 Gerador de Parsers LL(1) Dirigidos por Tabela

A tarefa de escrever um parser Top Down dirigido por tabela é bastante repetitiva e segue um padrão algorítmico, padrão esse que se quer captar, nesta proposta, na geração de código.
Assim, neste projeto deverás realizar as seguintes tarefas:

- Criar uma linguagem para descrição de gramáticas independentes de contexto;
- Criar um reconhecedor para essa linguagem que:
  1. Verifique se a linguagem é LL(1);
  2. Caso seja, constrói o parser Top-Down dirigido por tabela;
  3. Como extra, poderás pensar em arranjar uma maneira de juntar ações semânticas ao parser gerado.

Bom trabalho e boa sorte

A equipe docente:

José Carlos Ramalho
José João Almeida
Pedro Moura
Tiago Baptista