Texto auxiliar para Atividade Avaliativa #2: Analisador Léxico e Analisador Sintático

Pedro Santi Binotto [20200634][*1], Eduardo Caigar Dudel [18206076][†2], Arthur Alexandre Nascimento [18200410][‡3], and Eduardo Sousa Szczepaniak [20202478][§4]

[1]Departamento de Informática e Estatística, Universidade Federal de Santa Catarina

20 de julho de 2024

Resumo

Relatório do projeto prático da disciplina de *Introdução a Compiladores*.

[*]pedro.binotto@grad.ufsc.br
[†]dudeleduardo@gmail.com
[‡]arthuralexnascimento@hotmail.com
[§]szcz.eduardo@gmail.com

Contents

Elaboração do trabalho

O trabalho foi elaborando utilizando as ferramentas *Flex*[3] e *Bison*[1] para gerar os analisadores léxico e sintático, respectivamente. Além dos manuais das ferramentas, a apostila "*Lex & Yacc*"[2] foi outro recurso extensivamente consultado para utilização das tecnologias empregadas.

Para gerar o analisador sintático, uma descrição idêntica à especificação encontrada no documento da proposta do trabalho foi fornecida ao *Bison* para conversão.

Caso desejássemos construir uma *parse-table* LL(1), seria necessário adequar a escrita da sintaxe para possibilitar a análise *top-down sem backtracking*; entretanto, não é possível remover todos os casos de ambiguidades na gramática sem fazer alterações na especificação da linguagem, como a forma das clausulas "`if...else`" é inerentemente ambígua.

Uma versão da gramática ajustada para resolver os casos de ambiguidade possíveis e livre de recursão à esquerda pode ser encontrada na seção "Normalização da gramática".

A implementação final dos analisadores adiciona uma propriedade além do que descreve a especificação da gramática para possibilitar a escrita de comentários nos arquivos de código fonte:

```
/*
 * Texto dentro destes delimitadores não será interpretado.
 *
 * Palavras-chave e símbolos da linguagem também não produzem
 * efeito nenhum:
 *
 * def int print return () {} ;
 */

def main() {
  int A = calc_a();
  return;
}
```

Os arquivos de teste utilizam destes comentários como uma forma de registrar metadados sobre o caso de teste e saída esperada da execução da análise.

Normalização da gramática

Gramática original da linguagem

A seguir está a gramática original, como fornecida na proposta do trabalho, em notação BNF:

⟨*main*⟩ ::= ⟨*stmt*⟩
| ⟨*flist*⟩
| ⟨*empty*⟩

⟨*flist*⟩ ::= ⟨*fdef*⟩ ⟨*flist*⟩
| ⟨*fdef*⟩

⟨*fdef*⟩ ::= '`def`' '`id`' '`(`' ⟨*parlist*⟩ '`)`' '`{`' ⟨*stmtlist*⟩ '`}`'

⟨*parlist*⟩ ::= '`int`' '`id`' '`,`' ⟨*parlist*⟩
| '`int`' '`id`'
| ⟨*empty*⟩

⟨*stmt*⟩ ::= '`int`' '`id`' '`;`'
| ⟨*atribst*⟩ '`;`'
| ⟨*printst*⟩ '`;`'
| ⟨*returnst*⟩ '`;`'
| ⟨*ifstmt*⟩
| '`{`' ⟨*stmtlist*⟩ '`}`'
| '`;`'

⟨*atribst*⟩ ::= '`id`' = ⟨*expr*⟩
| '`id`' = ⟨*fcall*⟩

⟨*fcall*⟩ ::= '`id`' '`(`' ⟨*parlistcall*⟩ '`)`'

⟨*parlistcall*⟩ ::= '`id`' '`,`' ⟨*parlistcall*⟩
| '`id`'
| ⟨*empty*⟩

⟨*printst*⟩ ::= '`print`' ⟨*expr*⟩

⟨*returnst*⟩ ::= '`return`'

⟨*ifstmt*⟩ ::= '`if`' '`(`' ⟨*expr*⟩ '`)`' ⟨*stmt*⟩ '`else`' ⟨*stmt*⟩
| '`if`' '`(`' ⟨*expr*⟩ '`)`' ⟨*stmt*⟩

⟨*stmtlist*⟩ ::= ⟨*stmt*⟩ ⟨*stmtlist*⟩
| ⟨*stmt*⟩

⟨*expr*⟩ ::= ⟨*numexpr*⟩ '`<`' ⟨*numexpr*⟩
| ⟨*numexpr*⟩ '`>`' ⟨*numexpr*⟩
| ⟨*numexpr*⟩ '`==`' ⟨*numexpr*⟩
| ⟨*numexpr*⟩

⟨*numexpr*⟩ ::= ⟨*numexpr*⟩ '`+`' ⟨*term*⟩
| ⟨*numexpr*⟩ '`-`' ⟨*term*⟩
| ⟨*term*⟩

⟨*term*⟩ ::= ⟨*term*⟩ '`*`' ⟨*factor*⟩
| ⟨*factor*⟩

⟨*factor*⟩ ::= '`num`'
| '`(`' ⟨*numexpr*⟩ '`)`'
| '`id`'

Gramática ajustada

Gramática ajustada, livre de recursão à esquerda e corrigida para retificar todos os casos de ambiguidade que são possíveis; em notação BNF:

⟨*main*⟩ ::= ⟨*stmt*⟩
| ⟨*flist*⟩
| ⟨*empty*⟩

⟨*flist*⟩ ::= ⟨*fdef*⟩ ⟨*flist ′*⟩

⟨*flist ′*⟩ ::= ⟨*flist*⟩
| ⟨*empty*⟩

⟨*fdef*⟩ ::= '`def`' '`id`' '`(`' ⟨*parlist*⟩ '`)`' '`{`' ⟨*stmtlist*⟩ '`}`'

⟨*parlist*⟩ ::= '`int`' '`id`' ⟨*parlist ′*⟩

⟨*parlist ′*⟩ ::= '`,`' ⟨*parlist*⟩
| ⟨*empty*⟩

⟨*stmt*⟩ ::= '`int`' '`id`' '`;`'
| ⟨*atribst*⟩ '`;`'
| ⟨*printst*⟩ '`;`'
| ⟨*returnst*⟩ '`;`'
| ⟨*ifstmt*⟩
| '`{`' ⟨*stmtlist*⟩ '`}`'
| '`;`'

⟨*atribst*⟩ ::= '`id`' = ⟨*atribst ′*⟩

⟨*atribst ′*⟩ ::= ⟨*expr*⟩
| ⟨*fcall*⟩

⟨*fcall*⟩ ::= '`id`' '`(`' ⟨*parlistcall*⟩ '`)`'

⟨*parlistcall*⟩ ::= '`id`' ⟨*parlistcall ′*⟩

⟨*parlistcall ′*⟩ ::= '`,`' ⟨*parlistcall*⟩
| ⟨*empty*⟩

⟨*printst*⟩ ::= '`print`' ⟨*expr*⟩

⟨*returnst*⟩ ::= '`return`'

⟨*ifstmt*⟩ ::= '`if`' '`(`' ⟨*expr*⟩ '`)`' ⟨*stmt*⟩ ⟨*ifstmt ′*⟩

⟨*ifstmt ′*⟩ ::= '`else`' ⟨*stmt*⟩
| ⟨*empty*⟩

⟨*stmtlist*⟩ ::= ⟨*stmt*⟩ ⟨*stmtlist ′*⟩

⟨*stmtlist ′*⟩ ::= ⟨*stmtlist*⟩
| ⟨*empty*⟩

⟨*expr*⟩ ::= ⟨*numexpr*⟩ ⟨*expr ′*⟩

⟨*expr ′*⟩ ::= '`<`' ⟨*numexpr*⟩
| '`>`' ⟨*numexpr*⟩
| '`==`' ⟨*numexpr*⟩
| ⟨*empty*⟩

⟨*numexpr*⟩ ::= ⟨*term*⟩ ⟨*numexpr ′*⟩

⟨*numexpr ′*⟩ ::= '`+`' ⟨*term*⟩ ⟨*numexpr ′*⟩
| '`-`' ⟨*term*⟩ ⟨*numexpr ′*⟩
| ⟨*empty*⟩

⟨*term*⟩ ::= ⟨*factor*⟩ ⟨*term ′*⟩

⟨*term ′*⟩ ::= '`*`' ⟨*factor*⟩ ⟨*term ′*⟩
| ⟨*empty*⟩

⟨*factor*⟩ ::= '`num`'
| '`(`' ⟨*numexpr*⟩ '`)`'
| '`id`'

Referências Bibliográficas

References

[1] Charles Donnelly and Richard Stallman. Bison - the yacc-compatible parser generator. *Free Software Foundation*, 1992. Accessed: 2024-07-14.

[2] Tom Niemann. Lex & yacc. https://epaperpress.com/lexandyacc/. Accessed: 2024-07-14.

[3] Vern Paxson. Using flex - a fast lexical analyzer generator. *The Regents of the University of California*, 1990. Accessed: 2024-07-14.