Analisador Sintático

João Victor Cabral de Melo[1]

Universidade de Brasília

Abstract. This project is to show what we have done and learned in the course of compilers and the grammar of experimental language C-IPL.

Keywords: Compilers · Grammar · Lexical Analyser · Flex · Syntax Analyser · Bison

1 Introdução

1.1 Motivação

A maior motivação para a implementação de um compilador é conseguir gerar código executável em máquinas, dado uma linguagem de alto nível como entrada.

1.2 Nova Primitiva

A motivação para entrada da nova primitiva *list* é a possibilidade de resolver problemas onde se precisa de uma estrutura de dados com um tamanho não definido previamente. Facilitando esse conjunto específico de problemas pela não implementação da estrutura de dados manualmente.

2 Análise Léxica

A função da leitura do compilador é identificar lexemas na linguagem e passar uma sequência desse lexemas para o sintático avaliar se a ordem está correta. Neste projeto foi utilizado o **flex** [Pro12] para gerar o analisador léxico, além disso foram feitas tais modificações no arquivo, como as variáveis para contagem de linhas, colunas e erros léxicos além do tratamento de caso em que não consiga abrir o arquivo a ser escaneado, para ser entrada do usuário os lexemas a serem lidos. A tabela de símbolos será implementada como uma lista ligada, tendo como referência [ALSU06].

3 Análise Sintática

A função desta fase é verificar se as sentenças estão de acordo com a gramática. Como referência para construção da gramática C-IPL foi utilizado o [Hec21] e [Nal13]. Neste projeto foi utilizado o **bison** [RC21] como gerador sintático, além disso foram feitas modificações no arquivo como a inclusão de variável para contar o número de erros sintáticos e a opção de entrada do teclado do usuário em caso que não consiga abrir o arquivo dado como entrada do executável. A árvore sintática será implementada tendo como referência [ALSU06].

4 Testes

No diretório de testes possui dos arquivos com erros léxicos e sintáticos **error.c** e **error1.c**, sendo o segundo com uma *string* que não fechou aspas, e dois arquivos sem erros léxicos **success.c** e **success1.c**.

5 Instruções

Para se compilar o analisador sintático utiliza-se os seguintes comandos:

```
bison sintax.y -d -Wcounterexample
lex lexico.l
cc -Wall -Wextra -Wpedantic lex.yy.c sinttax.tab.c -o tradutor
./tradutor <caminho_do_arquivo_teste>
```

References

ALSU06. A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2 edition, 2006.

Hec21. R. Heckendorn. C- grammar, 2021. `http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf`. Acessado por último 02 Set 2021.

J.09. Levine J. *Flex and Bison*. O'reilly, 1 edition, 2009.

Mar21. J. D. Marangon. Compiladores para humanos, 2021. `https://johnidm.gitbooks.io/compiladores-para-humanos/content/`. Acessado por último 02 Set 2021.

Nal13. C. Nalon. Glc-precedencia, 2013.

Pro12. Flex Project. Flex Manual, 2012. `https://westes.github.io/flex/manual/index.html`.Acessado por último em 30 Ago 2021.

RC21. R. Stallman R. Corbett. Bison, 2021. `https://www.gnu.org/software/bison/manual/bison.html`. Acessado por último 30 Ago 2021.

SP21. IME SP. Tabela de precendência c, 2021. `https://www.ime.usp.br/~yw/bcc2000/C/tabela.html`.Acessado por último 02 Set 2021.

Appendices

Appendix A

Tabelas de Símbolos

Token	Informal Description	Sample Lexemes
ID	qualquer letra com '_' seguido de um número	jujuba_doce
LIST	qualquer tipo do list	list
TYPES	tipos int e float	int float
NULL	constante para listas vazias	NIL
STRING	qualquer caracter dentro de " "	"chocolate"
ASSIGN	Um único caractere =	=
KEYWORD	Palavras como if else for e return	if else for return
OUTPUT	Palavras da saída de dados da linguagem	write writeln
INPUT	Palavra de entrada de dados	read
CONSTRUCTOR	Palavras para construtores de lista	:
FUNCTION	Caracteres para as função de map e filter	>><<
COMMENT	Qualquer comentário em linha com // ou comentário multilinha com /**/	//pudim /* pudim é bom /
+ — - — * — /	Operações aritmeticas	+ - * /
—— — &&	Operações Lógicas	&& ——
>—<—>=—<=—==—!=	Operação de comparação	<<= >>= == !=
{}	Escopo	{}
()	Expressão	()
;	Separador de Expressão	;
?	Operação head da lista	?
!	Operação tail da lista ou negação	!
%	Operação tail destrutiva da lista	%
123	Número inteiro	123
.5	Número decimal	.5

Appendix B

Gramática

⟨program⟩ ::= ⟨paramlist⟩ (1)

⟨paramlist⟩ ::= ⟨param⟩ ⟨paramlist⟩ | ⟨param⟩ (2)

⟨param⟩ ::= ⟨variableParam⟩ | ⟨functionParam⟩ (3)

⟨variableParam⟩ ::= ⟨typeSpec⟩ ⟨var⟩ `;` (4)

⟨var⟩ ::= `ID` (5)

⟨typeSpec⟩ ::= ⟨TYPE⟩ | ⟨TYPE⟩ ⟨LISTTYPE⟩ (6)

⟨functionParam⟩ ::= ⟨typeSpec⟩ ⟨id⟩ `(` ⟨functionParams⟩ `)` ⟨stmt⟩ (7)

⟨functionParams⟩ ::= ⟨functionParamsList⟩ | ϵ (8)

⟨functionParamsList⟩ ::= ⟨functionParamsList⟩ `,` ⟨typeSpec⟩ `ID` (9)

| ⟨typeSpec⟩ `ID` (10)

⟨call⟩ ::= `ID (` ⟨functionParams⟩ `)` (11)

⟨stmList⟩ ::= ⟨stmList⟩⟨stmt⟩ | ⟨stmt⟩ (12)

⟨stmt⟩ ::= ⟨expStatement⟩ | ⟨compoundStatement⟩ (13)

| ⟨ifStatement⟩ | ⟨forStatement⟩ (14)

| ⟨returnStatement⟩ | ⟨inputStatement⟩ (15)

| ⟨outputStatement⟩ (16)

⟨expStatement⟩ ::= ⟨expression⟩ `;` (17)

⟨compoundStatement⟩ ::= { ⟨declaration stmList⟩ } (18)

⟨declaration⟩ ::= ⟨declaration⟩ ⟨variableParam⟩ (19)

| ϵ (20)

⟨ifStatement⟩ ::= `if (` ⟨expression⟩ `)` ⟨stmt⟩ (21)

| `if (` ⟨expression⟩ `)` ⟨stmt⟩ `else` ⟨stmt⟩ (22)

⟨forStatement⟩ ::= `for (` ⟨expStatement⟩⟨expStatement⟩ (23)

⟨expression⟩ `)`⟨stmt⟩ (24)

⟨returnStatement⟩ ::= `return` ⟨expression⟩ `;` (25)

⟨inputStatement⟩ ::= ⟨input⟩ `(` ⟨var⟩ `) ;` (26)

⟨outputStatement⟩ ::= ⟨output⟩ `(` ⟨term⟩ `) ;` (27)

⟨expression⟩ ::= `ID =` ⟨expression⟩ (28)

| ⟨orExpression⟩ (29)

⟨orExpression⟩ ::= ⟨orExpression⟩ `||` ⟨andExpression⟩ (30)

⟨andExpression⟩ ::= ⟨andExpression⟩ `&&` ⟨relationalExpression⟩ (31)
| ⟨relationalExpression⟩ (32)
⟨relationalExpression⟩ ::= ⟨relationalExpression⟩ ⟨REL_OP⟩ (33)
⟨listExpression⟩ (34)
| ⟨listExpression⟩ (35)
⟨listExpression⟩ ::= ⟨arithmExpression⟩ ⟨listOP⟩ (36)
⟨listExpression⟩ (37)
| ⟨arithmExpression⟩ (38)
⟨arithmExpression⟩ ::= ⟨arithmExpression⟩ ⟨SUB_ADD⟩ (39)
⟨airtmMulDivExpression⟩ (40)
| ⟨arihtmMulDivExpression⟩ (41)
⟨arithmMulDivExpression⟩ ::= ⟨arithmMulDivExpression⟩ ⟨MUL_DIV⟩ (42)
⟨term⟩ (43)
| ⟨term⟩ (44)
⟨term⟩ ::= ⟨const⟩ | ⟨call⟩ | `ID` (45)
| ⟨unaryTerm⟩ (46)
| ⟨immutable⟩ (47)
⟨immutable⟩ ::= `(` ⟨expression⟩ `)` (48)
⟨const⟩ ::= `INT` | `FLOAT` | `STRING` | `NIL` (49)
⟨listOP⟩ ::= ⟨FUNCTION⟩ | ⟨INFIX⟩ (50)
⟨output⟩ ::= `writeln` | `write` (51)
⟨input⟩ ::= `read` (52)
⟨TYPE⟩ ::= `int` | `float` (53)
⟨LISTTYPE⟩ ::= `list` (54)
⟨REL_OP⟩ ::= `<` | `<=` | `==` | `>` | `>=` (55)
⟨INFIX⟩ ::= `:` (56)
⟨FUNCTION⟩ ::= `>>` | `<<` (57)
⟨SUB_ADD⟩ ::= `-` | `+` (58)
⟨MUL_DIV⟩ ::= `*` | `/` (59)
(60)