Tradutor da Linguagem C-IPL

Pedro Vitor Valença Mizuno[17/0043665]

Universidade de Brasília, DF, Brasil

Resumo Neste documento é apresentado como foi criado o tradutor da linguagem C-IPL, uma linguagem baseada em C com algumas adições. O desenvolvimento do programa foi feito com auxilio da ferramenta FLEX, que permitiu uma maior facilidade na criação do analisador léxico, a primeira parte do tradutor.

Keywords: Analisador · Léxico · FLEX · Tradutor.

1 Motivação

Linguagens de programação são formas de se descrever, por meio de termos, uma computação de forma mais simples e clara para pessoas. No entanto, para que os comandos contidos nessa linguagem sejam compreendidos por uma máquina, é necessária a utilização de um tradutor, um programa que traduz as linhas de código para um formato capaz de ser executado pelo computador. [1]

Como projeto da disciplina Tradutores, foi proposta a criação de um tradutor para a linguagem C-IPL, uma linguagem de programação baseada em C. O C-IPL apresenta como seu maior diferencial a existência da primitiva *list*, que permite criar uma lista de inteiros ou números de ponto flutuante. A primitiva *list* é uma ferramenta poderosa devido ao fato de criar uma estrutura de dados em formato de lista que permite adicionar e remover elementos de forma simples. Além disso, as operações map e filter (>> e <<, respectivamente) geram uma forma efetiva de operar com funções sobre as listas utilizadas.

2 Descrição

A análise léxica consiste na primeira fase do tradutor, em que caracteres são agrupados a partir de regras definidas pela linguagem, resultando em lexemas. Como saída, o analisador léxico produz um *token*, o qual é composto por um nome e um valor, no formato (nome-token, valor-token), ou apenas o valor do *token* caso o nome seja equivalente ao valor, como palavras reservadas e símbolos como chaves e parênteses.

A partir do *software FLEX* [2], é possível criar expressões regulares para identificar os lexemas que compõem o programa em linguagem C-IPL, como tipos, identificadores, constantes, etc. Ao ocorrer uma correspondência entre lexema e expressão regular, é impresso o *token* correspondente e linha e coluna

onde ela ocorreu. Caso venha a ocorrer um erro léxico, ele é anunciado a partir de uma mensagem que aponta a linha e coluna onde houve o erro. Foram também tratados dois erros em específico, o comentário em múltiplas linhas sem fechamento e a string em múltiplas linhas. Detalhes acerca dos lexemas cobertos pelo analisador léxico estão presentes no Anexo A.

3 Arquivos Teste

A fim de testar as funcionalidades do analisador léxico, foram criados quatro arquivos de entrada em linguagem C-IPL, localizados no diretório *tests*, para serem avaliados pelo programa:

1. *teste1_correto.ci*: programa sem erros léxicos que recebe uma lista e retorna se a soma dos elementos contidos nessa lista é positiva ou negativa;
2. *teste2_correto.ci*: programa sem erros léxicos que monta em uma lista a sequência de *Fibonacci* até uma posição n;
3. *teste1_errado.ci*: programa com erros léxicos que calcula e apresenta a classificação do IMC de acordo com os dados inseridos. Os erros presentes são os pontos utilizados na linha 13, a string não fechada na linha 14, o *AND* com apenas um & e o formato do número de ponto flutuante na linha 17 e o comentário sem fechamento na linha 43;
4. *teste2_errado.ci*: programa com erros léxicos que encontra o menor número de ponto flutuante de uma lista. Os erros presentes são o formato do número de ponto flutuante da linha 3, strings sem fechamento nas linha 14 e 30, o símbolo barra invertida na linha 25 e o símbolo . na linha 34.

4 Instrução de Compilação

Versões das ferramentas utilizadas:

- Ubuntu 21.04
- FLEX 2.6.4
- GCC 11.1.0
- Make 4.3
- Kernel 5.11.0-25-generic

Para a facilitar a compilação do analisador léxico criado, foi criado um *makefile* que sintetiza os passos com apenas um comando. Por meio de um terminal, no diretório do projeto (i.e. 17_0043665_lexico), onde está o arquivo *makefile*, execute o comando: **make NAME=*arquivo_teste.ci***. Após sua execução será apresentada no terminal a análise léxica obtida do teste utilizado.

Referências

1. Aho, Alfred V., et al.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc, USA (2006)
2. Repositório do Flex, https://github.com/westes/flex. Último acesso em 8 de agosto de 2021

A Léxico

Tabela 1. Tabela que apresenta o léxico da linguagem C-IPL.

Lexema	Nome do Token	Valor do Atributo
Espaço	-	-
Comentário	-	-
if	***if***	-
else	***else***	-
for	***for***	-
return	***return***	-
read	***read***	-
write	***write***	-
writeln	***writeln***	-
Identificador	***IDENTIFICADOR***	Apontador para a entrada da tabela
Constante	***CONSTANTE***	Apontador para a entrada da tabela
int	***TIPO***	int
float	***TIPO***	float
int list	***TIPO***	int list
float list	***TIPO***	float list
+	***ARITOP***	+
-	***ARITOP***	-
*	***ARITOP***	*
/	***ARITOP***	/
&&	***LOGOP***	&&
||	***LOGOP***	||
<=	***RELOP***	<=
<	***RELOP***	<
>=	***RELOP***	>=
>	***RELOP***	>
==	***RELOP***	==
! =	***RELOP***	! =
<=	***RELOP***	<=
:	***LISTOP***	:
?	***LISTOP***	?
!	***LISTOP***	!
%	***LISTOP***	%
>>	***LISTOP***	>>
<<	***LISTOP***	<<

Tabela 2. Segunda parte da tabela que apresenta o léxico da linguagem C-IPL.

Lexema	Nome do Token	Valor do Atributo
(	(	-
)	)	-
{	**{**	-
}	**}**	-
[	[	-
]	]	-
,	**,**	-
;	**;**	-

B Gramática

A gramática utilizada no tradutor da linguagem C-ILP é apresentada nas seguintes expressões:

1. *programa* → *inicio*
2. *inicio* → *inicio declarar* | *declarar*
3. *declarar* → *funcao* | *variavel*;
4. *funcao* → **TIPO IDENTIFICADOR** (*parametros*) {*interior*}
5. *TIPO* → **int** | **float** | **int list** | **float list**
6. *parametros* → *parametro* , *parametros* | ∈
7. *parametro* → **TIPO IDENTIFICADOR**
8. *interior* → *variavel*; | *condicional* | *iteracao* | *expressao*; | *chamaFunc* | *IO*; | *return*
9. *IO* →)*entrada*| *saida*
10. *entrada* → **READ**(**IDENTIFICADOR**)
11. *saida* → **WRITE**(**CONSTANTE**) | **WRIETLN**(**CONSTANTE**) | **WRITE** (**IDENTIFICADOR**) | **WRIETLN**(**IDENTIFICADOR**)
12. *variavel* → **TIPO IDENTIFICADOR**
13. *condicional* → **IF** (*expLogic*) {*interior*} | **IF** (*expLogic*) {*interior*} **ELSE** {*interior*}
14. *iteracao* → **FOR**(*inicialIte*; *testeIte*; *atualizaIte*) {*interior*}
15. *inicialIte* → *expressao* | ∈
16. *testeIte* → *expLogic* | ∈
17. *atualizaIte* → *expressao* | ∈
18. *expressao* → **IDENTIFICADOR** = *expLogic* | **IDENTIFICADOR** = *listaOP*
19. *listaOP* → !**IDENTIFICADOR** | ?**IDENTIFICADOR** | %**IDENTIFICADOR** | **IDENTIFICADOR** << **IDENTIFICADOR** | **IDENTIFICADOR** >> **IDENTIFICADOR**
20. *expLogic* → *expComp* && *expLogic* | *expComp* ||*expLogic*| !*expLogic*
21. *expComp* → *expArit* **RELOP** *expArit* | *expArit*
22. *RELOP* → < | <= | == | ! = | > | >=
23. *expArit* → *expArit* + *expMul* | *expArit* - *expMul* | *expMul*

24. *expMul* → *expMul* * *elemento* | *expMul* / *elemento* |
25. *elemento* → **IDENTIFICADOR** | (*expLogic*) | *chamaFunc* | *CONSTANTE*
26. *chamaFunc* → **IDENTIFICADOR** (*argumentos*)
27. *argumentos* → *argumento*, *argumentos* | ∈
28. *argumento* → **IDENTIFICADOR** | *expLogic*
29. *return* → **RETURN CONSTANTE**; | **RETURN IDENTIFICADOR**;