UNIVERSIDADE FEDERAL DE UBERLÂNDIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CONSTRUÇÃO DE COMPILADORES
PROJETO DA DISCIPLINA

Arthur do Prado Labaki
11821BCC017

Uberlândia - MG
2022

Sumário

1 Descrição do Projeto

O projeto da disciplina consiste no desenvolvimento das etapas de um *front-end* de um compilador para a linguagem simplificada demonstrada mais abaixo. Todas as imagens, códigos, diagramas e qualquer outros arquivos utilizados neste trabalho estão inseridos em meu repositório publico no GitHub, o link de acesso está disponível abaixo.
Repositório do projeto

Cada capítulo deste relatório se trata de uma etapa da criação do projeto completo, totalizando quatro etapas. Na primeira etapa, será feito as especificações da linguagem, como a criação da gramática livre de contexto, como também a identificação dos *tokens* e suas respectivas expressões regulares. Já na segunda parte, será feito a análise léxica, especificando o diagrama de transição e a implementação manual do analisador léxico.
Na terceira parte, será realizada a análise sintática, fazendo ajustes na gramática, criando grafos e também implementando o analisador sintático preditivo. Por fim, na quarta e última etapa, será realizado a tradução dirigida por sintaxe, efetuando a análise semântica, gerando o código intermediário, além de realizar algumas etapas programadas, tudo disponível no slide do projeto.

Especificação da linguagem para o projeto:

```
Estrutura principal:
programa nome_programa bloco

Bloco:
begin declaração das variáveis sequência de comandos end

Declaração de variáveis:
tipo: lista_ids ;
Sendo os tipos: int, char e float

Comando de seleção:
if(cond) then bloco else bloco

Comentários:
[ texto_comentário ]
```

```
Comandos de repetição:
while(cond) do bloco    ou
repeat bloco while(cond)

Comando de atribuição:
id:= expressão ;

Condições:
Igual (=), diferente (~=), menor (<), maior (>),
menor ou igual (<=), maior ou igual (>=)

Expressões:
Soma (+), subtração (-), multiplicação (*),
divisão (/) e exponenciação (^)
char deve estar entre apóstrofo (ex: 'A')
int deve estar entre -32768 e +32767
float pode ser ponto fixo (ex: 5.3) ou
notação científica (ex: 0.1E-2)
parênteses para priorizar operações
```

2 Projeto da Linguagem

Nesta etapa, é feito a especificação da linguagem. Para isso, primeiro é necessário definir a gramatica livre de contexto (GLC), com as estruturas da linguagem especificada.

G=({EP, Bloco, Dvar, Scom, Var, IDs, Com, Cond, Expr, Term}, {programa, ID, begin, end, tipo, : , ; , (,) , := , if, then, else, while, do, repeat, relop, ariop, num, ϵ}, P, EP), sendo:
P = {
EP -> **programa** ID Bloco
Bloco -> **begin** DVar SCom **end**
DVar -> Var | Var DVar | ϵ
SCom -> Com | Com SCom | ϵ
Var-> **tipo :** IDs **;**
IDs -> ID | ID , IDs
Com -> **if** (Cond) **then** Com **else** Com | ID **:=** Expr **;** | Expr **;** | ϵ |
| **while** (Cond) **do** Com | **repeat** Com **while** (Cond)
Cond -> Expr **relop** Expr
Expr -> Expr **ariop** Term | Term
Term-> num | ID | (Expr)
}

Após a definição da gramática, é realizado a identificação dos *tokens* usados na gramatica. Para isso, é oportuno utilizar uma tabela para colocar todos os *tokens* já identificados, com os seus respectivos lexemas e valores.

Lexemas	Nome do Token	Valor do atributo
Qualquer número	num	Posição na tabela de símbolos
Qualquer ID	ID	Posição na tabela de símbolos
int	tipo	INT
char	tipo	CHAR
float	tipo	FLOAT
programa	programa	-
begin	begin	-
end	end	-
if	if	-
then	then	-
else	else	-
while	while	-
do	do	-
repeat	repeat	-
:	:	-
;	;	-
(	(	-
)	)	-
:=	:=	ATR
=	relop	EQ
~=	relop	NE
<	relop	LT
>	relop	GT
<=	relop	LE
>=	relop	GE
+	ariop	ADD
-	ariop	SUB
*	ariop	MUL
/	ariop	DIV
^	ariop	EXP
Qualquer ws	-	-
Qualquer comentário	-	-

Tabela 1 – Tabela dos Tokens

Para a conclusão dessa etapa do projeto, é definido os padrões, em expressão regular, para os lexemas aceitos em cada *token*.

digito → [0 - 9]
dígitos → digito (digito)*
num → dígitos ("."dígitos) ? (E[+ -] ? dígitos) ?
letra_ → [A - Za - z_]
ID → letra_ (letra_ | digito)*
tipo → int | char | float
programa → programa

begin → begin
end → end
if → if
then → then
else → else
while → while
do → do
repeat → repeat
: → :
; → ;
(→ (
) →)
:= → :=
relop → = | ~= | < | > | <= | >=
ariop → + | - | * | / | ^
ws → (“ “ | \t | \n)
comentário → "[". "]"

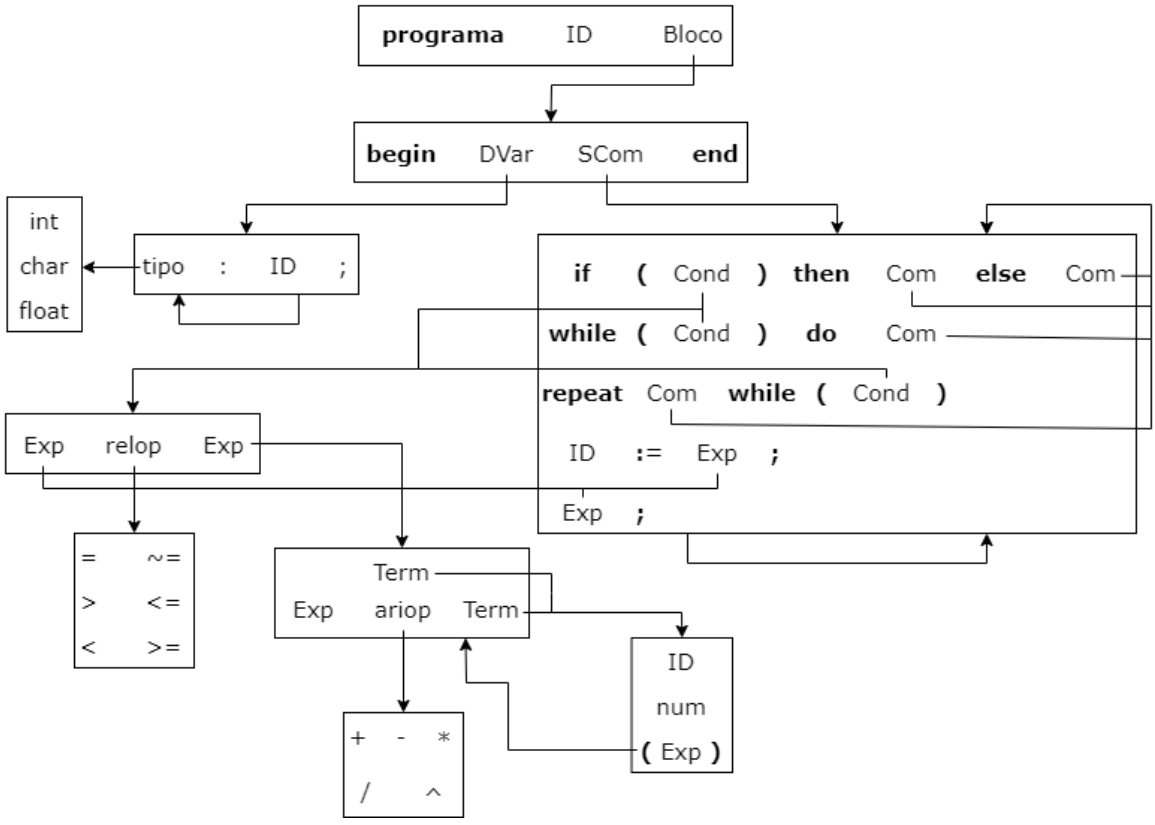

Figura 1 – Diagrama para facilitar entendimento da Linguagem

3 Análise Léxica

4 Análise Sintática

5 Tradução Dirigida por Sintaxe