
**UNIVERSIDADE FEDERAL DA FRONTEIRA SUL**
**CURSO CIÊNCIA DA COMPUTAÇÃO**

**EVERALDO PEREIRA GOMES**
**NATANAEL HENRIK ZAGO**

# RELATÓRIO DA CONSTRUÇÃO DE COMPILADOR

**CHAPECÓ**
**2021**

# RESUMO

O presente trabalho tem como objetivo aplicar e demonstrar as técnicas aprendidas durante o desenvolvimento da CCR de Construção de Compiladores para a implementação do projeto de um compilador para uma linguagem própria. Este compilador contempla as etapas de análise léxica e sintática, ficando as demais etapas para a construção de um compilador completo pendentes.

Para o desenvolvimento do compilador foi utilizado a linguagem de programação C++ utilizando os paradigmas de orientação a objeto, também foi utilizado como apoio o programa GoldParser para geração da tabela LALR utilizada na análise sintática.

# INTRODUÇÃO

Desde o advento da computação diversas formas para programar os computadores foram desenvolvidos, passando desde técnicas de cartões perfurados até chegar nas linguagens de alto nível (semelhante a linguagem humana) que possuímos hoje em dia. Para ser possível usar as linguagens de alto nível foi necessário a construção de técnicas para fazer com que os computadores entendessem as linguagens de alto nível transformando em linguagens de máquina, em um desses processos entram os compiladores.

O processo de compilação consiste em traduzir um código fonte escrito em uma linguagem geralmente mais aproximada da linguagem humana para um código objeto onde um computador é capaz de entender. Durante o processo de compilação são feitas diversas etapas passando pela análise léxica, sintática e semântica depois para ser gerado um código intermediário, após isso é construído o código para ser de fato interpretado pelo computador.

# REFERENCIAL TEÓRICO

## Analise Lexica

A análise léxica é a primeira etapa feita em um processo de compilação, nesta etapa o código fonte escrito pelo programador vai ser quebrado em partes separando as palavras em tokens, onde a partir da interpretação dos tokens é verificando se eles pertencem ao dicionário da linguagem. No processo de análise léxica é gerada a saída para as próximas etapas de compilação e também os erros léxicos do programa escrito.

## Analise Sintatica

A análise sintática ocorre logo após a etapa de análise léxica, esta verifica se a sequência de tokens possui uma ordem válida para a linguagem, estas geralmente são definidas através de uma GLC (Gramática livre de contexto). A saída da análise sintática acusa se houve algum erro na escrita do programa.

# IMPLEMENTAÇÕES E RESULTADOS

## Definições das características das linguagem, tokens e GLC

**- Declaração de variáveis:**

*Símbolo:* Deve conter apenas letras x ou y (Não podem ser palavras reservadas)

*Explicação:* Declaração de variáveis.

*Exemplo:* xy = 1 fa ou xx = 0 fa

**- Declaração de números:**

*Símbolo:* Deve conter apenas letras 0 ou 1

*Explicação:* Declaração de números.

*Exemplo:* 01 ou 11

**- Atribuição:**

*Símbolo:* =

*Explicação:* Atribuir valor às variáveis.

*Exemplo:* x = 01 fa ou y = 01 fa

**- Operadores matemáticos:**

*Símbolo:* +

*Explicação:* somar variáveis ou números.

*Exemplo:* x = 1 + a fa ou y = 10 + 1 fa
*Símbolo:* -

*Explicação:* subtrair variáveis ou números.

*Exemplo:* x = 1 - a fa ou y = 10 - 10 fa

*Símbolo:* *

*Explicação:* multiplicação de variáveis ou números.

*Exemplo:* x = 1 * a fa ou y = 10 * 10 fa

*Símbolo:* /

*Explicação:* divisão de variáveis ou números.

*Exemplo:* x = 1 / a fa ou y = 10 / 1 fa

**- Operadores condicionais:**

*Símbolo:* eq

*Explicação:* comparar se variáveis ou números são iguais.

*Exemplo:* 1 eq 1 ou 1 eq a
*Símbolo:* n

*Explicação:* comparar se variáveis ou números são diferentes.

*Exemplo:* 1 n 1 ou 1 n x

*Símbolo:* me

*Explicação:* comparar se variáveis ou números é menor ou igual a outro(a).

*Exemplo:* y me x ou x me y

*Símbolo:* ma

*Explicação:* comparar se variáveis ou números é maior ou igual a outro(a).

*Exemplo:* y ma x ou x ma y

**- Estrutura de condição:**

**Onde:* COND é a condição a ser verificada
e BLOCO bloco de código a ser executado.

*Síntax:* se *COND ie*
BLOCO
es

*Explicação:* a palavra reservada *se* inicia a estrutura, depois segue a verificação da condição (COND), após vem *código a ser executado)*, seguindo, se condição verdadeira executa o código (BLOCO) do *se*, o fim da estrutura do condicional termina com a palavra reservada *es.*

*Exemplo:* se *y eq 100 ie*
y = y + 01 fa
es

**- Estrutura de repetição:**

**Onde:* COND é a condição a ser verificada
e BLOCO bloco de código a ser executado

*Síntax:* enq *COND fa es*
BLOCO
fe

*Explicação:* a palavra reservada *enq* inicia a estrutura depois vem a condição a ser verificada (COND), segue-se para o bloco de código (BLOCO) a ser executado caso condição seja verdadeira, a estrutura termina com a palavra reservada *fe.*

*Exemplo:* enq *y me 100 es*
x = x +1 fa
y = y + 10 fa
fe

*Exemplo 2:* enq *y ma 100 es*
x = x + 1 fa
y = y - 10 fa
fe

## Tokens:

| | |
|---|---|
| se | #Início condicional |
| fs | #Fim condicional |
| ie | #Início código executável do condicional |
| is | #Início código executável do laço de repetição |
| enq | #Início repetição |
| fe | #Fim repetição |
| fa | #Fim atribuição |
| + | #Soma (+) |
| - | #Subtração (-) |
| * | #Multiplicação (*) |
| / | #Divisão (/) |
| eq | #Igualdade (==) |
| n | #Diferença (!=) |
| ma | #Maior igual (>=) |
| me | #Menor igual (<=) |
| = | #Atribuição (=) |

## Gramática Livre de Contexto

"start symbol" = <S>
<S> ::= <OPS>
<OPS> ::= <OPC> <OPS> | <OPR><OPS> | <ATR><OPS> | ε
<OPC> ::= 'se' <CQE> 'ie' <OPS> 'fs'
<OPR> ::= 'enq' <CQE> 'is' <OPS> 'fe'

<ATR> ::= <VAR> '=' <OPA> 'fa' | <VAR> '=' <VAR> 'fa' | <VAR> '=' <NUM> 'fa'

<CQE> ::= <VAR> <EQI> <VAR> | <VAR> <EQI> <NUM> | <NUM> <EQI> <VAR> |
<NUM> <EQI> <NUM>
<EQI> ::= 'eq' | 'n' | 'me' | 'ma'

<OPA> ::= <VAR> <OPE> <VAR> | <VAR> <OPE> <NUM> | <NUM> <OPE>
<VAR> | <NUM> <OPE> <NUM>
<OPE> ::= '+' | '-' | '*' | '/'

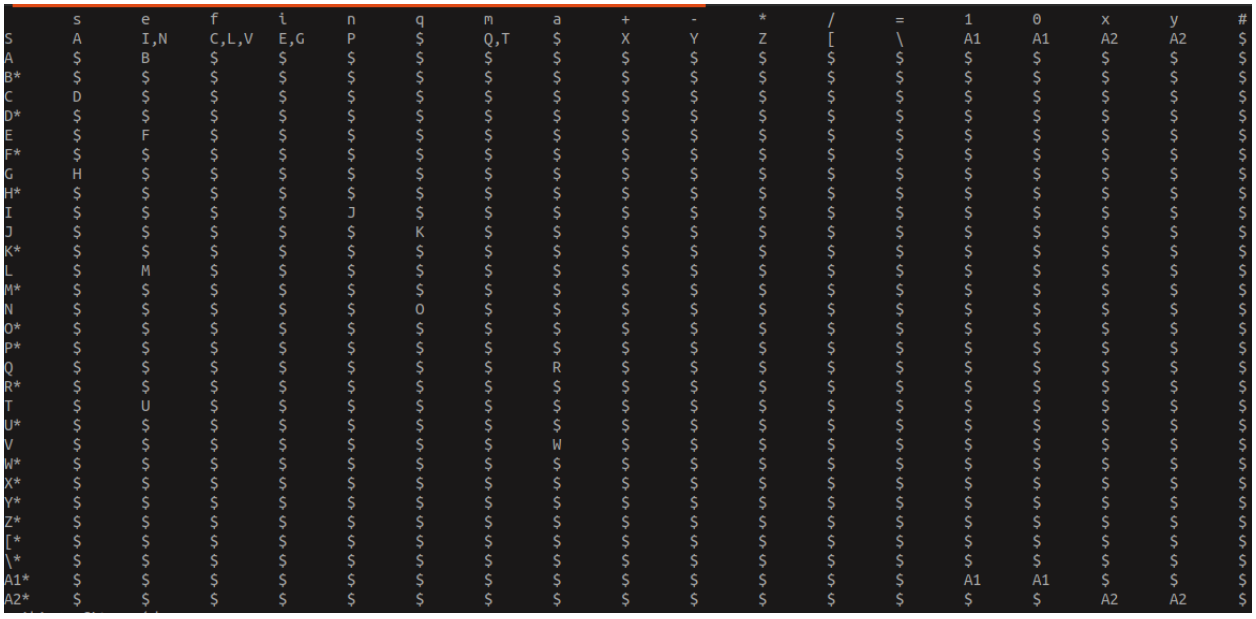
<NUM> ::= '0'<N> | '1'<N>
<N>  ::= '0'<N> | '1'<N> | Ɛ

<VAR> ::= 'x'<A> | 'y'<A>
<A>  ::= 'x'<A> | 'y'<A> | Ɛ

## Etapas da Construção do Compilador

### 1) Etapa construção autômato

A primeira etapa para a construção do nosso compilador se consistiu na criação de um autômato finito não determinístico (AFN) contendo os estados para a interpretação dos tokens da linguagem. Abaixo temos a ilustração do resultado final do autómato.



| | s | e | f | i | n | q | m | a | + | - | * | / | = | 1 | 0 | x | y | # |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | A | I,N | C,L,V | E,G | P | $ | Q,T | $ | X | Y | Z | [ | \ | A1 | A1 | A2 | A2 | $ |
| A | $ | B | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| B* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| C | D | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| D* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| E | $ | F | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| F* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| G | H | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| H* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| I | $ | $ | $ | $ | J | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| J | $ | $ | $ | $ | $ | K | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| K* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| L | $ | M | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| M* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| N | $ | $ | $ | $ | $ | O | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| O* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| P* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| Q | $ | $ | $ | $ | $ | $ | $ | R | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| R* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| T | $ | U | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| U* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| V | $ | $ | $ | $ | $ | $ | $ | W | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| W* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| X* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| Y* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| Z* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| [* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| * | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ |
| A1* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | A1 | A1 | $ | $ | $ |
| A2* | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | $ | A2 | A2 | $ |

## 2) Etapa analisador léxico

Na etapa de análise léxica foi utilizado o autômato da etapa anterior para a interpretação do arquivo escrito pelo programador. Neste foram quebrados em tokens e estes eram verificados conforme os estados do autômato, verificando se era estado de aceitação ou de possíveis erros. Como saída desta etapa foi gerado uma fita com os identificadores dos tokens para utilização para as próximas etapas e também foram gerados a saída de erros sintáticos do programa escrito.

## 3) Etapa analisador sintático

Nesta etapa foi utilizado a fita de saída da análise léxica bem como a tabela LALR gerada pela ferramenta de apoio GoldParser da GLC da linguagem criada. A partir da tabela LALR e da fita de saída com os identificadores da análise léxica foi implementado as ações da LALR, esta consistia em verificar se seria feito um empilhamento, uma redução ou se era um estado de aceitação.

# CONCLUSÕES

Durante o desenvolvimento do trabalho nós podemos ver em prática como funciona alguma das técnicas e os algoritmos para a construção de um compilador, além de demonstrar o funcionamento da análise léxica e sintática.

As principais dificuldades encontradas foram a escolha da linguagem de programação para a implementação, onde começamos o trabalho com a linguagem C, mas no decorrer tivemos que mudar para o C++ devido ao grande uso de strings que nos facilitou. Outra dificuldade foi aprender a utilizar o GoldParser e interpretar a saída por ele gerada, e fazer o nosso programa ler e interpretar essa saída. Também nos faltou tempo para concluir todas as etapas propostas, apesar de termos trabalhado com bastante afinco no trabalho.

A implementação do nosso programa ficou de fácil entendimento e visível a aplicação das técnicas para a construção de compiladores, um dos fatores que ajudou foi utilizar a programação orientada a objetos do C++. Como pendências, o programa em alguns casos não estava aceitando devidas entradas onde não conseguimos identificar o erro.

O desenvolvimento do trabalho foi muito proveitoso, onde nos acrescentou muito conhecimento a respeito de compiladores onde tivemos que ver a parte teórica e aplicar na prática. Também a criação de uma sintaxe para nossa linguagem foi muito interessante. Recomendamos este trabalho para as próximas turmas.

## REFERENCIAS


WIKIPÉDIA, A ENCICLOPÉDIA LIVRE.. **Wikipédia, a enciclopédia livre..** *[S.l.].* Wikipédia, a enciclopédia livre., 2020. Disponível em: https://pt.wikipedia.org/wiki/Compilador#Hist%C3%B3ria. Acesso em: 12 out. 2021.

JOHNI DOUGLAS MARANGON. **GitBook.** *[S.l.].* GitBook, 2020. Disponível em: https://johnidm.gitbooks.io/compiladores-para-humanos/content/. Acesso em: 12 out. 2021.