INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO MATO GROSSO

Vitor Bruno de oliveira barth

**RELATÓRIO FINAL DE ATIVIDADE ACADÊMICA – IMPLEMENTAÇÃO DE COMPILADOR**

Cuiabá – MT

2018

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO MATO GROSSO

Campus Cuiabá

Departamento de Área de Informática

vitor bruno de oliveira barth

**RELATÓRIO FINAL DE ATIVIDADE ACADÊMICA – IMPLEMENTAÇÃO DE COMPILADOR**

Relatório Final de Atividade Acadêmica, referente a implementação de um compilador, como parte dos requisitos necessários para a conclusão da disciplina Compiladores do curso de Engenharia de Computação do Campus Cuiabá do Instituto Federal do Mato Grosso.

Professor da Disciplina: Dr. Ed’ Wilson Tavares Ferreira

Cuiabá – MT

2018

**RESUMO**

Neste documento é apresentado o relato da experiência de implementação de um compilador, desenvolvido na disciplina de Compiladores. Foi proposta uma gramática que possui recursos básicos de uma linguagem de programação. Todas as fases de um compilador foram desenvolvidas, e optou-se em gerar o código final para a linguagem *Assembly* compatível com o montador MASM, de modo que fosse possível realmente executar os *softwares* implementados na linguagem criada.

Palavras-chave: compiladores, *assembly*, *TypeScript*.

**LISTA DE ILUSTRAÇÕES**

Figura 1 – Modelo de Compilador 10

Figura 2 - Modelo Esquemático do Autômato de Estados Finitos 13

Figura 3 - Erro Léxico 15

Figura 4 - Erro Sintático 16

Figura 5 - Erro Semântico 16

Figura 6 - Programa que lê de soma dois números 17

Figura 7 - Programa que verifica se um número é primo 17

**LISTA DE TABELAS E QUADROS**

Quadro 1 – Lista de Tokens de Comandos 21

Quadro 2 – Lista de Tokens de Lógica e Matemática 21

Quadro 3 – Símbolos Não Terminais 22

Quadro 4 – Lista de Construções 22

**LISTA DE SIGLAS**

GALS – Gerador de Analisadores Léxicos e Sintáticos

MASM – *Microsoft Macro Assembler*

**SUMÁRIO**

INTRODUÇÃO 10

1 Metodologia 11

2 Objetivos 12

2.1 Objetivo Geral 12

2.2 Objetivos Específicos 12

3 Gramática 12

4 Autômato 13

5 Características do Compilador 14

6 Resultados Alcançados 15

7 Exemplos de Uso 15

8 Dificuldades Encontradas 18

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 19

REFERÊNCIAS 20

Apêndice I – Definição da Gramática 21

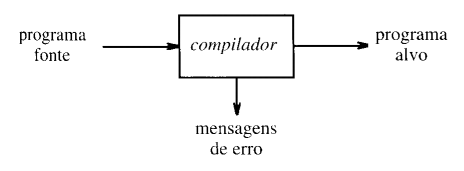
# INTRODUÇÃO

Linguagens de programação são ferramentas utilizadas para comunicar instruções à um computador (DERSHEM, 1995), e possuem diversas regras para que possa ser devidamente interpretada e executada por um computador. O conjunto de palavras que está de acordo com estas regras constitui o código fonte de um *software* (AHO, 1986), i. e., um algoritmo que o computador deve seguir para realizar a tarefa idealizada (DERSHEM, 1995).

Um compilador é um *software* que transforma um código fonte em um código final, ou seja, converte uma linguagem mais próxima da linguagem humana em instruções que são compreendidas e executadas pelos computadores (MARANGON, 2017). Durante esta conversão são detectados erros no código fonte, que devem ser informados ao seu usuário (AHO, 1986). A Figura 1 exemplifica este modelo de Compilador.

Fonte: AHO, 1986.

Figura – Modelo de Compilador



Através da evolução das Linguagens de Programação – que está diretamente ligada à evolução dos Compiladores – o desenvolvimento e manutenção de programas é facilitado: tarefas que antes eram responsabilidade do programador, tais como gerenciamento de memória, verificação de tipos e execução de tarefas em paralelo, o que permite o aumento da produtividade dos desenvolvedores, diminuindo tempo de desenvolvimento e os custos ligados diretamente à esta atividade (MARANGON, 2017).

Apesar da sua importância para a computação, as tarefas que um compilador executa para atingir seu objetivo podem ser divididas em apenas dois grupos: análise e síntese (AHO, 1986).

A análise identifica dentro do código fonte estruturas que fazem parte da gramática linguagem que está sendo compilada – identificando estruturas inválidas no código fonte – e cria uma interpretação intermediária do mesmo (AHO, 1986; RICARTE, 2008).

Após todo o código ser verificado livre de erros, é possível convertê-lo à linguagem de destino. Contudo, não é possível geralmente realizar uma conversão direta entre a linguagem fonte e a linguagem alvo, sendo utilizada a representação da estrutura original criada durante a fase de análise (RICARTE, 2008).

Este trabalho mostra o desenvolvimento de um Compilador capaz de interpretar um código fonte de nova linguagem de programação, chamada ‘Bob’, e gerar um código final que é reconhecido pelo do Sistema Operacional *Microsoft Windows 10.* As seções são divididas como segue: a seção 1 descreve a metodologia utilizada no desenvolvimento do compilador, a seção 2 descreve os objetivos propostos, a seção 3 descreve a gramática da linguagem desenvolvida interpretada pelo compilador, a seção 4 descreve o autômato utilizado pelo compilador, a seção 5 descreve tecnicamente o compilador desenvolvido, a seção 6 descreve os resultados alcançados, a seção 7 mostra alguns exemplos de programas utilizando a linguagem Bob e a seção 8 descreve as dificuldades encontradas durante o desenvolvimento,

# Metodologia

Inicialmente foi desenvolvida a gramática da linguagem Bob. Utilizou-se o *software* GALS (Gerador de Analisadores Léxicos e Sintáticos) (UFSC, 2003), que avalia a gramática através do algoritmo LL(1).

Após a gramática possuir todos os requisitos propostos – apresentados na seção 4 – e ser avaliada através do algoritmo LL(1) como uma Gramática Livre de Contexto, foi criado um modelo esquemático do autômato utilizado pelo compilador por meio do *software* JFLAP (ROGER, 2009).

Terminada a modelagem da gramática, deu-se início à implementação. Foi utilizada a linguagem de programação *TypeScript* 2.6.1, interpretada pelo Ambiente de Execução *Node.js* 8.9.2.

O compilador Bob foi desenvolvido em quatro fases distintas: analisador léxico, analisador sintático, analisador semântico, gerador de código intermediário e gerador de código final. Cada uma destas fases será descrita na seção 5.

Como resultado, foi gerado um código fonte em linguagem *Assembly*, reconhecida pelo montador *Microsoft Macro Assembler* 11 (MICROSOFT, 2000), que gera um arquivo executável compatível com o sistema operacional *Windows* 10 *Education*,versão 1709 (MICROSOFT, 2018).

# Objetivos

## Objetivo Geral

* Desenvolver um compilador capaz de transpilar a linguagem Bob para linguagem *Assembly* compatível com o montador MASM.

## Objetivos Específicos

* Criar uma Gramática Livre de Contexto que atenda aos requisitos do algoritmo LL(1);
* Criar um compilador capaz de analisar léxica, sintática e semanticamente a gramática desenvolvida, e gerar uma saída em linguagem *Assembly* compatível com o montador MASM;
* Compreender o funcionamento de um compilador ao aplicar a teoria aprendida em sala de aula durante a disciplina de Compiladores.

# Gramática

A linguagem Bob foi modelada para atender os seguintes requisitos:

* Possuir ao menos um tipo de variável;
* Possuir comandos de entrada via teclado e saída via console;
* Possuir ao menos uma estrutura condicional completa (Se… Então… Senão)
* Possuir ao menos uma estrutura de repetição;
* Reconhecer as quatro operações aritméticas básicas (adição, subtração, multiplicação e divisão) em expressões com qualquer quantidade de operandos;
* Realizar as quatro operações lógicas básicas (maior que, menor que, igual a e diferente de) em expressões com dois operandos;
* A gramática não deve ter recursão à esquerda e deve estar em sua forma fatorada, além de não ser ambígua (i. e., passar nos testes do algoritmo LL(1)).

Como resultado, foi desenvolvida uma Gramática que:

* Possui com 25 *tokens*, 17 símbolos não terminais e 34 expressões;
* Realiza operações números reais positivos;
* Reconhece blocos aninhados;
* Reconhece blocos de texto (caracteres alfanuméricos sem acentuação) para impressão em tela;
* Permite a declaração de variáveis em qualquer lugar do código fonte;
* Não diferencia o escopo para declaração de variáveis
* É interpretada *top-down* (de cima para baixo, da esquerda para a direita)
* Não possuir recursão à direita, ambiguidade e está em sua forma fatorada, ou seja, é uma Gramática Livre de Contexto do tipo LL(1).

A lista de *tokens* e a definição formal da gramática está presente no Apêndice I.

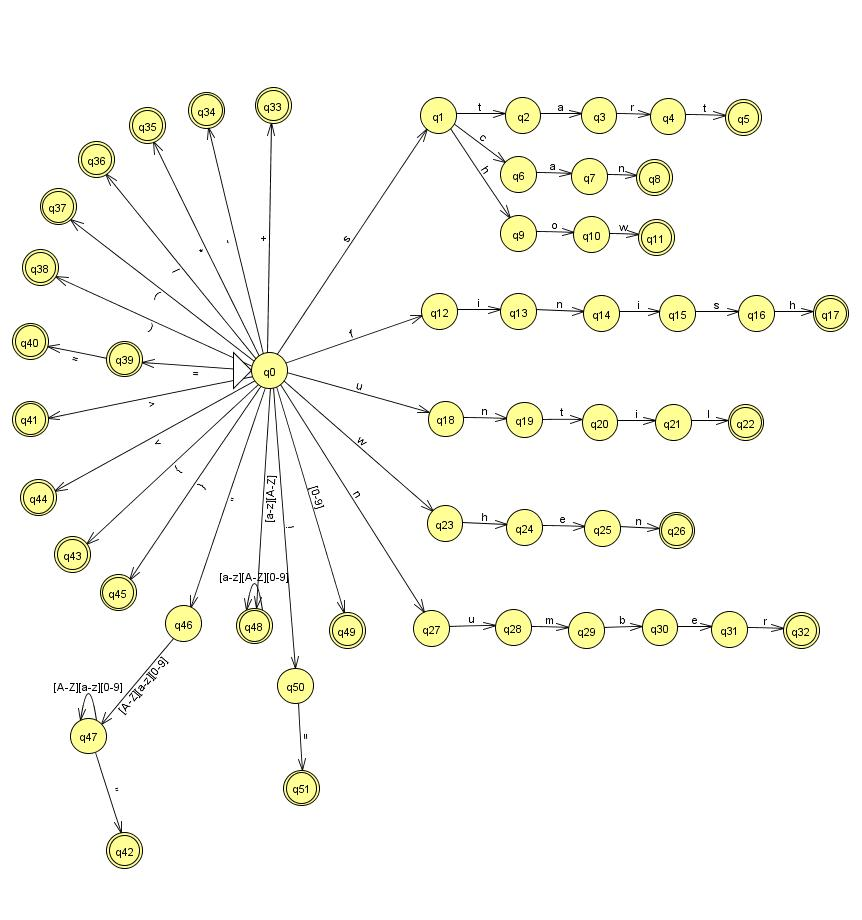
# Autômato

A identificação das expressões da gramática utilizou um autômato finito.

Um autômato finito é um modelo matemático que está em somente um estado de cada vez e armazena informações sobre estados passados. Com ele é possível identificar os *tokens* digitados por meio da leitura sequencial de cada caractere do código fonte.

O modelo esquemático que define os estados que o autômato pode estar é mostrado na Figura 2.

Figura - Modelo Esquemático do Autômato de Estados Finitos



Fonte: Elaborado pelo Autor

# Características do Compilador

O compilador foi desenvolvido na linguagem de programação TypeScript e é dividido quatro módulos:

1. Analisador Léxico: Recebe o código fonte como uma única *string* contínua, a qual é avaliada por uma máquina de estado finito, a qual identifica cada *token* e o salva em uma tabela de *tokens*. Nesta fase são identificados erros como a utilização de caracteres inválidos, como caracteres acentuados e caracteres não alfanuméricos.
2. Analisador Sintático: Recebe a tabela de *tokens*, e busca atingir expressões terminais através das construções que fazem parte da definição formal da gramática da linguagem desenvolvida. Nesta fase são identificados a maior parte dos erros, como a utilização de *tokens* em posições incorretas, chamadas a comandos inexistentes, falta de demarcadores de arquivo, de fim de linha, de bloco, etc.
3. Analisador Semântico: Através da tabela de *tokens*, o Analisador Semântico verifica a se todas as variáveis utilizadas foram devidamente declaradas, verifica os tipos e ainda se há alguma operação matemática inválida (tal como divisão explícita por zero). Após verificar erros, é gerada uma tabela de variáveis que contém informações sobre todas as variáveis utilizadas pelo código fonte.
4. Gerador de Código Intermediário: Esta fase possui executa funções distintas. A primeira é a limpeza da tabela de *tokens*: após o código fonte ser verificado sem erros nas três etapas de análise, *tokens* de controle de início e fim de programa, fim de linha e de declaração de variáveis e tipos não são mais necessários. Sabendo que um processador consegue apenas realizar operações com dois operandos, operações complexas são divididas em operações com no máximo dois operandos. E por fim, a terceira função divide operações lógicas em dois tokens: um de comparação e outro de salto.
5. Geração de Código Final: Recebe o código intermediário e a tabela de variáveis. É responsável por converter os comandos da linguagem fonte em comandos da linguagem de destino. Além disso, é responsável por criar códigos para alocação de espaço em memória para variáveis, e chamar bibliotecas do Sistema Operacional responsáveis pela escrita e leitura no console.

# Resultados Alcançados

O compilador desenvolvido é capaz de transpilar um código fonte em linguagem Bob para um código final em linguagem *Assembly* compatível com o montador MASM. Foi desenvolvido totalmente em *TypeScript* e atende a todos os requisitos propostos.

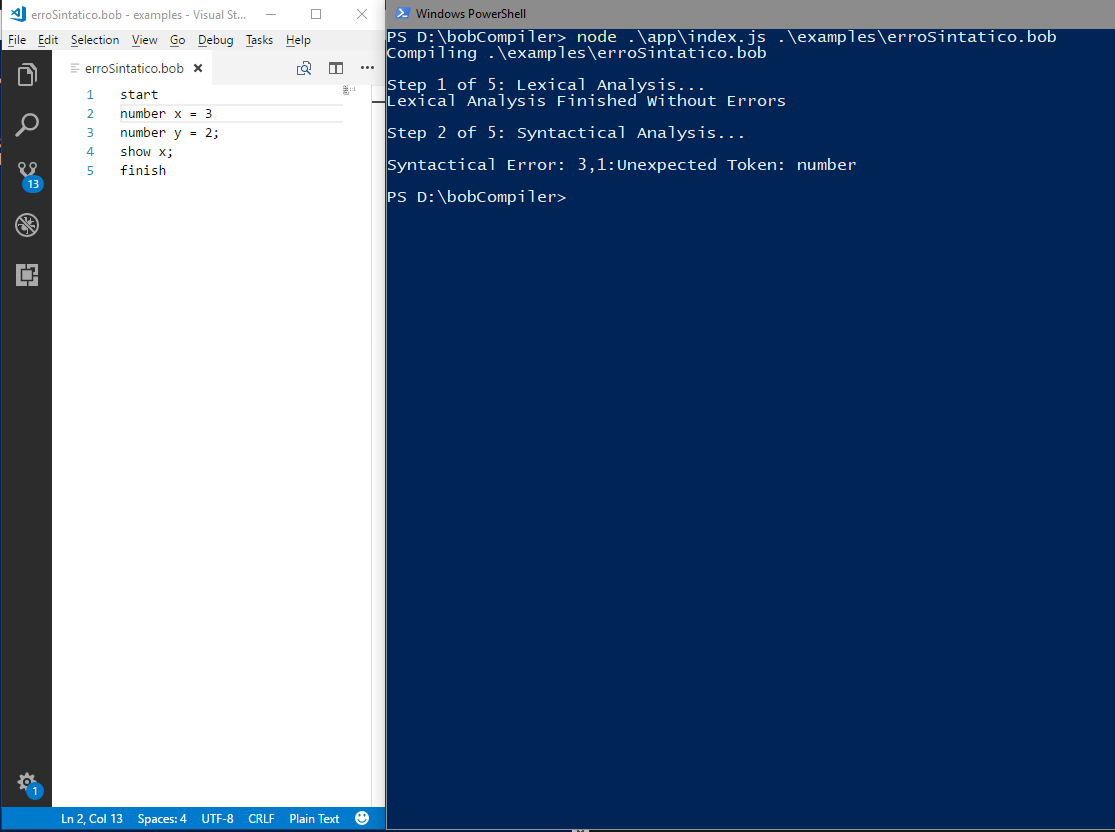
Implementar um compilador permitiu compreender e desmistificar o funcionamento das linguagens de programação, e de seus respectivos compiladores.

# Exemplos de Uso

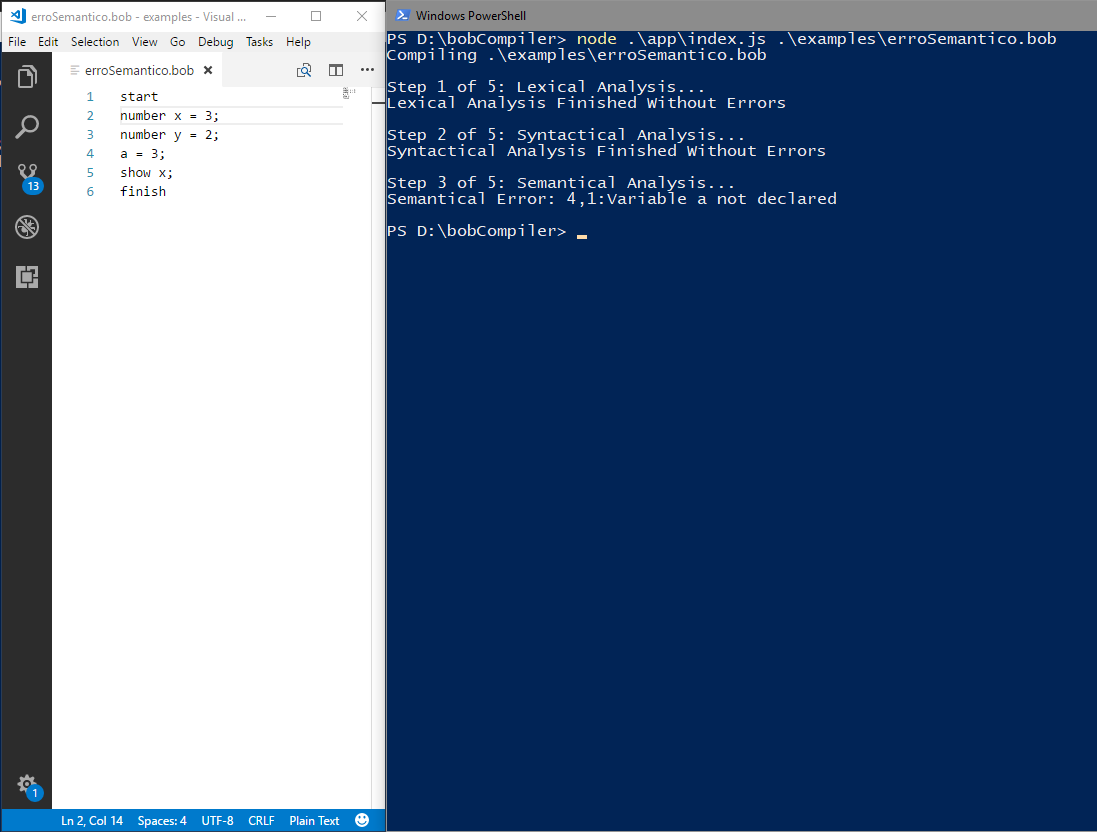
Figura 3 - Erro Léxico

Fonte: Elaborado pelo Autor

Figura 4 - Erro Sintático

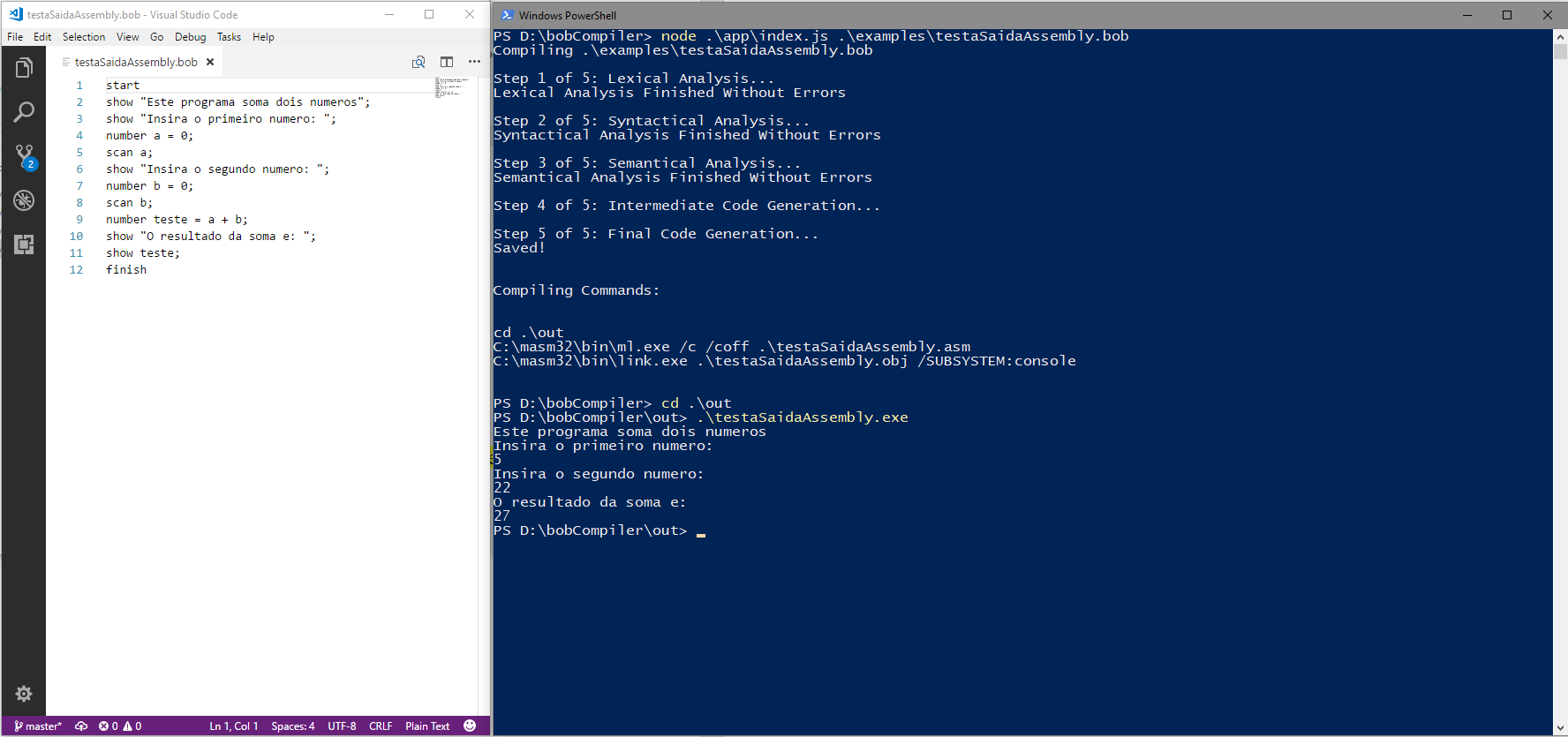


Fonte: Elaborado pelo Autor

Figura 5 - Erro Semântico

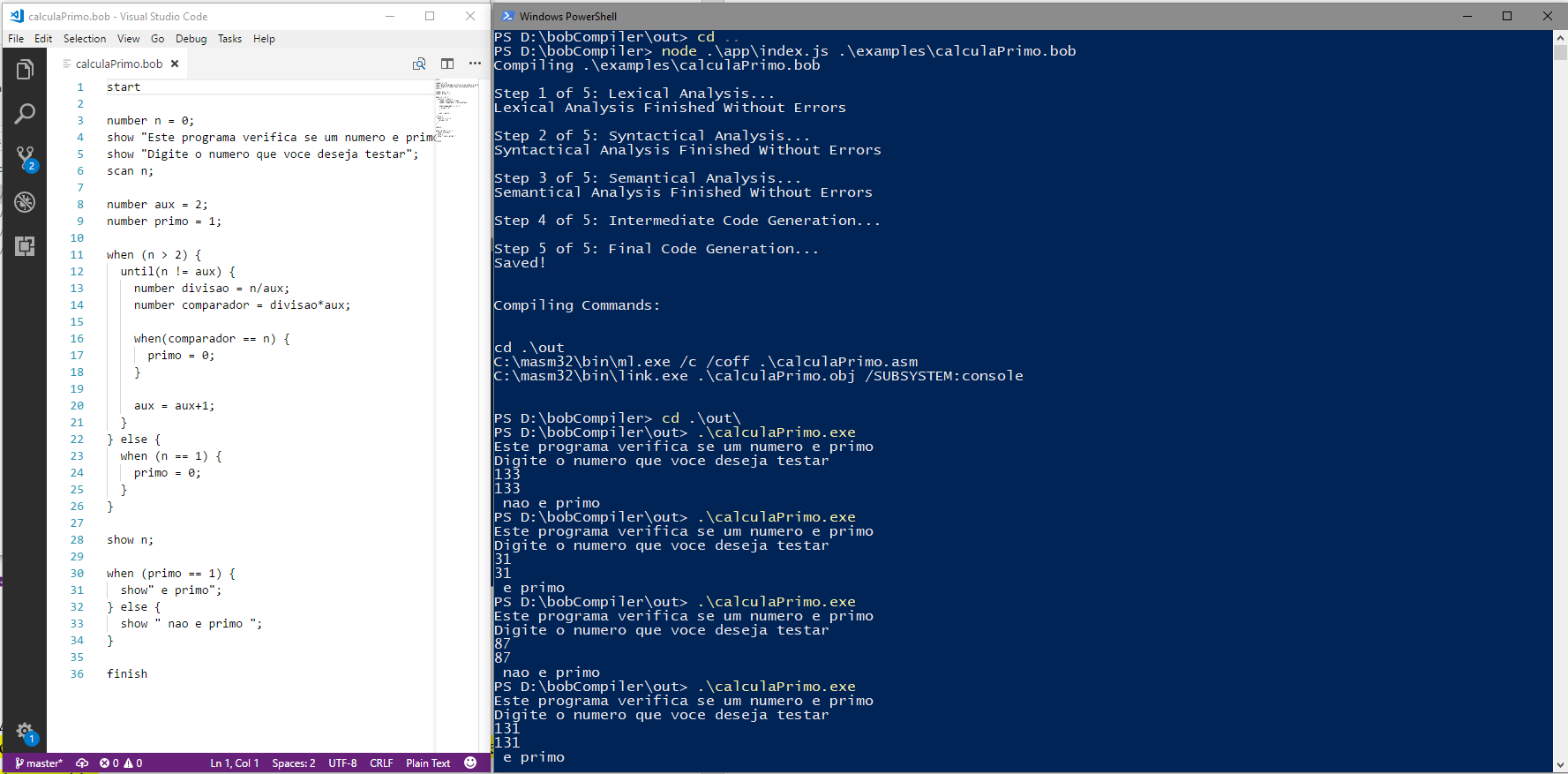
Fonte: Elaborado pelo Autor

Figura 6 - Programa que lê de soma dois números



Fonte: Elaborado pelo Autor

Figura - Programa que verifica se um número é primo



Fonte: Elaborado pelo Autor

# Dificuldades Encontradas

O desenvolvimento deste Compilador permitiu utilizou de conhecimentos adquiridos durante o todo o decorrer do curso. Conhecimentos das disciplinas Algoritmos, Estruturas de Dados, Linguagens de Programação, Arquitetura de Computadores, Sistemas Operacionais, Microprocessadores foram necessários para que se pudesse atingir o objetivo proposto.

Além de conhecimentos, perseverança foi essencial. É extremamente trabalhoso cansativo desenvolver o mesmo *software* por meses à fio. Conforme o código cresce, a manutenção e a depuração se tornam mais complexa o tem de se esperar que todo o resto esteja funcionando como deve.

Contudo, o resultado final: ver um *software* em execução que foi escrito em uma linguagem singular, própria, é uma experiência que vale à pena.

# CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Os conhecimentos obtidos durante o desenvolvimento deste projeto foram essenciais para que se percebesse a ligação entre as disciplinas do curso de Engenharia da Computação. Além disso, a prática, neste caso, aumentou a compreensão da teoria vista durante a disciplina de Compiladores.

Mas nada é perfeito: o compilador desenvolvido ainda possui alguns *bugs.* Algumas mensagens de erro não estão de acordo com a realidade, e determinados *tokens* não estão tão funcionais quanto deviam. Portanto, pretende-se otimizar e corrigir o código do compilador de modo que este possa servir de modelo para turmas futuras.

A linguagem *TypeScript* mostrou-se eficaz, e algumas de suas funcionalidades, sobretudo operações sobre Listas e *Strings,* facilitaram o desenvolvimento do Compilador proposto.

# REFERÊNCIAS

AHO, Alfred V. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Publishing Company. 1986.

DERSHEM, Herbert L.; Jipping, Michael J. *Programming Languages. Structures and Models*. PWS Publishing Company. 1995

MARANGON, Johni Douglas. Compiladores para Humanos. Gitbooks. 2017.

MICROSOFT. MASM32 SDK 11.2000. Disponível em: <http://www.masm32.com>. Acesso em: 26 jan. 2018.

MICROSOFT. Windows 10 Education. 2018. Disponível em: <https://www.microsoft.com/pt-br/software-download/windows10>. Acesso em: 21 fev. 2018.

RICARTE, Ivan. Introdução à Compilação. Elsevier. 2008.

ROGER, Susan H. *JFLAP*. Duke University, 2009. Disponível em: <http://www.jflap.org>. Acesso em: 25 set. 2017.

UFSC – Universidade Federal de Santa Catarina. Gerador de Analisadores Léxicos e Sintáticos. 2003. Disponível em: <http://gals.sourceforge.net>. Acesso em: 20 set. 2017.

# Apêndice I – Definição da Gramática

Quadro – Lista de *Tokens* de Comandos

|  |  |  |
| --- | --- | --- |
| **Token** | **Lexema** | **Descrição** |
| tk\_inicio | start | Denota o início do Código Fonte |
| tk\_fim | finish | Denota o fim do Código Fonte |
| tk\_escrever | show | Comando para escrever no console |
| tk\_ler | scan | Comando para ler entrada do teclado |
| tk\_repeticao | until | Comando de repetição (Faça até que X condição seja atingida) |
| tk\_condicao\_se | when | Comando de condição (Caso a condição X seja verdadeira, faça) |
| tk\_condicao\_senao | else | Comando de exceção (Caso a condição X seja falsa, faça) |
| tk\_final\_linha | ; | Denota o final de uma expressão |
| tk\_chaves\_abre | { | Denota o início de um bloco de código |
| tk\_chaves\_abre | } | Denota o fim de um bloco de código |

Fonte: Elaborado pelo Autor

Quadro – Lista de *Tokens* de Lógica e Matemática

|  |  |  |
| --- | --- | --- |
| **Token** | **Lexema** | **Descrição** |
| tk\_atibuicao | = | Indica uma atribuição de valores para uma variável |
| tk\_matematica\_soma | + | Operador Matemático: Soma |
| tk\_matematica\_subtracao | - | Operador Matemático: Subtração |
| tk\_matematica\_multiplicacao | \* | Operador Matemático: Multiplicação |
| tk\_matematica\_divisao | / | Operador Matemático: Divisão |
| tk\_logica\_menor | < | Operador Lógico: Menor que |
| tk\_logica\_maior | > | Operador Lógico: Maior que |
| tk\_logica\_igualdade | = = | Operador Lógico: Igual a |
| tk\_logica\_diferente | != | Operador Lógico: Diferente de |
| tk\_parenteses\_abre | ( | Abre-Parênteses |
| tk\_parenteses\_fecha | ) | Fecha-Parênteses |

Fonte: Elaborado pelo Autor

Quadro – Símbolos Não Terminais

|  |  |
| --- | --- |
| **Identificação** | **Descrição** |
| <PROGRAMA> | Demarca o programa |
| <LISTA\_COMANDOS> | Demarca *tokens* de comando e seus operandos |
| <COMANDO> | Demarca *tokens* de comando |
| <ARGUMENTO\_GERAL> | Demarca os argumentos das expressões de comando (caso haja) |
| <ARGUMENTO\_STRING> | Demarca argumentos do tipo texto |
| <ARGUMENTO\_NUMERICO> | Demarca argumentos do tipo numérico |
| <MATEMATICA> | Demarca uma operação matemática (com operadores e operandos) |
| <EXPRESSAO\_ARITMETICA> | Diferencia os operandos dos operadores de uma expressão matemática |
| <OP\_MATEMATICA> | Demarca os *tokens* operadores matemáticos em uma expressão matemática |
| <LOGICA> | Demarca uma operação lógica (com operadores e operandos) |
| <OP\_LOGICA> | Demarca os *tokens* operadores lógicos em uma operação lógica |
| <EXPRESSAO\_LOGICA> | Diferencia os operandos dos operadores de uma expressão lógica |
| <DECLARACAO> | Demarca uma operação de declaração de variável nova |
| <ATRIBUICAO> | Demarca uma operação de atribuição de valores à uma variável (nova ou existente) |
| <ALTERAR\_VALOR> | Demarca a variável, o *token* de atribuição e o valor recebido em uma operação de atribuição |
| <SENAO> | Demarca o *token* de comando de exceção |

Fonte: Elaborado pelo Autor

Quadro – Lista de Construções

|  |
| --- |
| ESTRUTURA GERAL DO PROGRAMA |
| <PROGRAMA> ::= tk\_inicio <LISTA\_COMANDOS> tk\_fim |
| <LISTA\_COMANDOS> ::= <COMANDO> <LISTA\_COMANDOS> |
| <LISTA\_COMANDOS> ::= î |
| COMANDOS |
| <COMANDO> ::= tk\_escrever <ARGUMENTO\_GERAL> tk\_final\_linha |
| <COMANDO> ::= tk\_ler tk\_id tk\_final\_linha |
| <COMANDO> ::= tk\_repeticao <LOGICA> tk\_chaves\_abre <LISTA\_COMANDOS> tk\_chaves\_fecha |
| <COMANDO> ::= tk\_condicao\_se <LOGICA> tk\_chaves\_abre <LISTA\_COMANDOS> tk\_chaves\_fecha <SENAO> |
| <COMANDO> ::= <DECLARACAO> tk\_final\_linha |
| <COMANDO> ::= <ALTERAR\_VALOR> tk\_final\_linha |
| VERIFICAÇÃO DE SINTAXE - LÓGICA |
| <LOGICA> ::= tk\_parenteses\_abre <EXPRESSAO\_LOGICA> tk\_parenteses\_fecha |
| <EXPRESSAO\_LOGICA> ::= <MATEMATICA> <OP\_LOGICA> <MATEMATICA> |
| <SENAO> ::= tk\_condicao\_senao tk\_chaves\_abre <LISTA\_COMANDOS> tk\_chaves\_fecha |
| <SENAO> ::= î |
| VERIFICAÇÃO DE SINTAXE - MATEMÁTICA |
| <MATEMATICA> ::= <ARGUMENTO\_NUMERICO> <EXPRESSAO\_ARITMETICA> |
| <MATEMATICA> ::= tk\_parenteses\_abre <MATEMATICA> tk\_parenteses\_fecha <EXPRESSAO\_ARITMETICA> |
| <EXPRESSAO\_ARITMETICA> ::= <OP\_MATEMATICA> <MATEMATICA> |
| <EXPRESSAO\_ARITMETICA> ::= î |
| OPERADORES LÓGICOS E MATEMÁTICOS |
| <OP\_MATEMATICA> ::= tk\_matematica\_soma |
| <OP\_MATEMATICA> ::= tk\_matematica\_subtracao |
| <OP\_MATEMATICA> ::= tk\_matematica\_multiplicacao |
| <OP\_MATEMATICA> ::= tk\_matematica\_divisao |
| <OP\_LOGICA> ::= tk\_logica\_menor |
| <OP\_LOGICA> ::= tk\_logica\_maior |
| <OP\_LOGICA> ::= tk\_logica\_igualdade |
| <OP\_LOGICA> ::= tk\_logica\_diferente |
| VERIFICAÇÃO DE TIPOS |
| <ARGUMENTO\_GERAL> ::= <ARGUMENTO\_STRING> |
| <ARGUMENTO\_GERAL> ::= <MATEMATICA> |
| <ARGUMENTO\_NUMERICO> ::= var\_numerico |
| <ARGUMENTO\_NUMERICO> ::= tk\_id |
| <ARGUMENTO\_STRING> ::= tk\_texto |
| DECLARAÇÃO E ATRIBUIÇÃO |
| <DECLARACAO> ::= tk\_tipo\_numerico tk\_id <ATRIBUICAO> |
| <ATRIBUICAO> ::= tk\_atribuicao <MATEMATICA> |
| <ATRIBUICAO> ::= î |
| <ALTERAR\_VALOR> ::= tk\_id tk\_atribuicao <MATEMATICA> |

Fonte: Elaborado pelo Autor