INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO MATO GROSSO

PEDRO FELIPE GONÇALVES DE ARRUDA

**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

PEDRO FELIPE GONÇALVES DE ARRUDA

**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 proposto uma gramática que possui recursos básicos de uma linguagem de programação. Todas as fases de um compilador foram desenvolvidas, porém optou-se em gerar um código final em *Assembly* por meio do NASM para sistema operacional *Windows* 32 *bits*, tendo em vista a execução da linguagem em um aplicação real.

Palavras-chave: Compiladores, *Assembly, Python*

**ABSTRACT**

This document presents a compiler implementation experience, that was developed through Compiler discipline. A grammar with basic programming language features was proposed. All compiler phases were developed, but instead it has been decided to generate a final code in Assembly through NASM for 32-bit Windows operating system, in order to execute the language in a real application.

Keywords: Compilers, Assembly, Python

**LISTA DE ILUSTRAÇÕES**

[Figura 1 - Compilador 10](#_Toc532814694)

[Figura 2- Autômato 15](#_Toc532814695)

[Figura 3– Erro léxico 16](#_Toc532814696)

[Figura 4– Erro sintático 17](#_Toc532814697)

[Figura 5– Erro Semântico 17](#_Toc532814698)

[Figura 6– Fitorial de um número 18](#_Toc532814699)

**LISTA DE TABELAS**

[Tabela 1– Tabelade tokens 12](#_Toc532814712)

[Tabela 2– Tabela de produções 13](#_Toc532814713)

**LISTA DE SIGLAS**

GALS – Gerador de Analisadores Léxicos e Sintáticos

NASM – *Netwide Assembler*

**SUMÁRIO**

[INTRODUÇÃO 10](#_Toc532803681)

[1. Etapas do PROJETO 11](#_Toc532803682)

[1.1. Metodologia 11](#_Toc532803683)

[1.2. Objetivo Geral 11](#_Toc532803684)

[1.3. Objetivos Específicos 11](#_Toc532803685)

[1.4. Gramática 12](#_Toc532803686)

[1.5. Autômato 15](#_Toc532803687)

[1.6. Fases do Compilador 15](#_Toc532803688)

[1.8. Exemplos de Uso 16](#_Toc532803689)

[CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 19](#_Toc532803690)

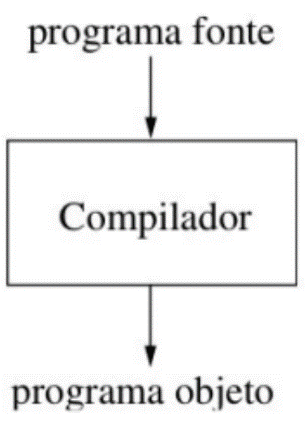
[REFERÊNCIAS 20](#_Toc532803691)

**INTRODUÇÃO**

O mundo como conhecemos depende das linguagem de programção, pois muiotas pessoas usam os computadores no dia a dia, e com isso usam varios programas que foram escritos em alguma linguagem de programação. Mas antes que possa ser execultados , um programa deve ser traduzido para um formato que pemita ser excutado por um computador. O programa que faz a tradução é conhecido como compilador.(AHIO, 2008).

De fomra simples, o compilador é um programa que traduiz um programa escrito em uma linguagem de programação (programa fonte) que recebe com entrada, em um programa equivalente em outra linguagem(programa objeto). Tendo como um dos principais papeis retratar erros no programa fonte detectados durante a tradução.

Figura 1 - Compilador



Fonte: AHO,2008

Com o avanço da tecnologico ao passar do tempo, os computadores ficarm mais acessiveis e necessarios. Assim havendo cada vez a necessidade de melhoramento para que se conseguise atender o mercado, desta forma fazendo com que houvesse o desenvolvimento de novos programas para computadores.

Assim tendo a necessidade de criação de novas linguagens de programação, visando cada vez facilitando a vida do programador, cada vez podendo criar novos programas sem ter um conhecimento aprofundado a arquitetura de desenvolvimento. Com essas necessidades os Compiladores foram evoluindo para acompalhar e melhorar o desenvolvimentos das linguagens de programação e o sugimento de novas. Assim passando a geerar codigo para diversas plataformas diferentes.

Diante disto, esse relatorio descreve o desenvolvimento de um compilador passo a passo, comtodas as funções basicas necessarias para o funcionamento do mesmo, desenvolvido pelo autor na disciplina de compiladores do curso de Engenharia da Computação.

# Etapas do PROJETO

## Metodologia

A primeira etapa foi desenvolvimento do compilador, foi elaborar a gramática de uma linguagem de programação, ultilizando-se da ferramenta de *software* livre: GALS (Gerador de Analisadores Léxicos e Sintáticos) (UFSC, 2003).

Na segunda etapa após o termino da criação da gramática, foi criado um autômato da gramática com o ultilização do *software* JFLAP (ROGER, 2009), para auxíliar no conhecimento da sintaxe da linguagem.

Na terceira etapa foi implementado o compilador, ultilizando – se a linguagem de programação *Python* na versão 3.6.6 em ambiente *Windows*  por meio da IDE PyCharm.

Na ultima etapa foi gerar o programa objeto em *Assembly* compativel com o montador NASM.

## Objetivo Geral

* Desenvolver uma linguagem que a sua gramática seja capaz de declarar um tipo de variavel, possua comandos de entrada e saída de dados, controle de fluxo e de iteração.
* Costruir um compilar capaz de traduizir a linguagem desenvolvida para a linguagem *Assembly* para o NASM no sistema operacional *Windows.*

## Objetivos Específicos

* Criar criada uma Gramática Livre de Contexto que atenda aos requisitos do algoritmo LL(1) com o auxílio da ferramenta GALS.
* Desenvolver o compilador com todas as usas etapas:
  + Analise léxica;
  + Analise Sintática;
  + Analise Semântica;
  + Geração de Código.
* Executar algoritmos na gramática desenvolvida para serem traduzidas pelo compilador.

## Gramática

* A linguagem foi desenvoda de forma que atenda os seguintes requisitos:
  + Declarar pelo menos um tipo de variavel;
  + Possuir comandos de entrada e saida de dados;
  + Possuir controle de fluxo ou condiciona(Se... Senão...);
  + Possuir ao menos uma estrutura de repetição;
  + Reconhecer e calcular expressões matematicas com as quatros oprações basicas(adição, subtração, multiplicação e divisão);
  + Realizar as seis operações logicas (maior igual que, maior que,menor igual que, menor que, igual e diferente);
* Baseando nos requisitos, foi desenvolvido uma gramática que:
  + Possui 29 *tokens*, 18 síbolos não terminais, 30 expressões;
  + Realiza operações numerias inteiras;
  + Reconheca texto alfanuméricos leitura do teclado e impressão em tela
  + É uma gramática livre de contexto do tipo LL(1), não possui recursão à direita, ambiguidade e está fatorada, e interpretada *top-down* (de cima pra baixo, da esquerda para a direita).

Tabela 1– Tabelade tokens

|  |  |
| --- | --- |
| **Token** | **Lexema** |
| inicio | begin |
| fim | end |
| escrever | output |
| condicao\_se | if |
| senao | @ |
| condicao\_repeticao | loop |
| fim\_repeticao | lend |
| atribuicao | <- |
| logica\_maior\_que | >= |
| logica\_menor\_que | <= |
| logica\_igual | == |
| logica\_maior | > |
| logica\_menor | < |
| logica\_diferente | != |
| matematica\_soma | + |
| matematica\_subtracao | - |
| matematica\_multiplicacao | \* |
| matematica\_divisao | / |
| matematica\_igual | = |
| final\_linha | ; |
| dois\_pontos | : |
| ponto\_interrogacao | ? |
| abre\_parenteses | ( |
| fecha\_parenteses | ) |
| virgula | , |
| tipo | int |
| variavel | {L}({L}|{D})\* |
| texto | \"({L}|{D}|{WS}|{S}|{SIMB})\*\" |
| numero | {D}\* |

Fonte: Elaborado pelo Autor

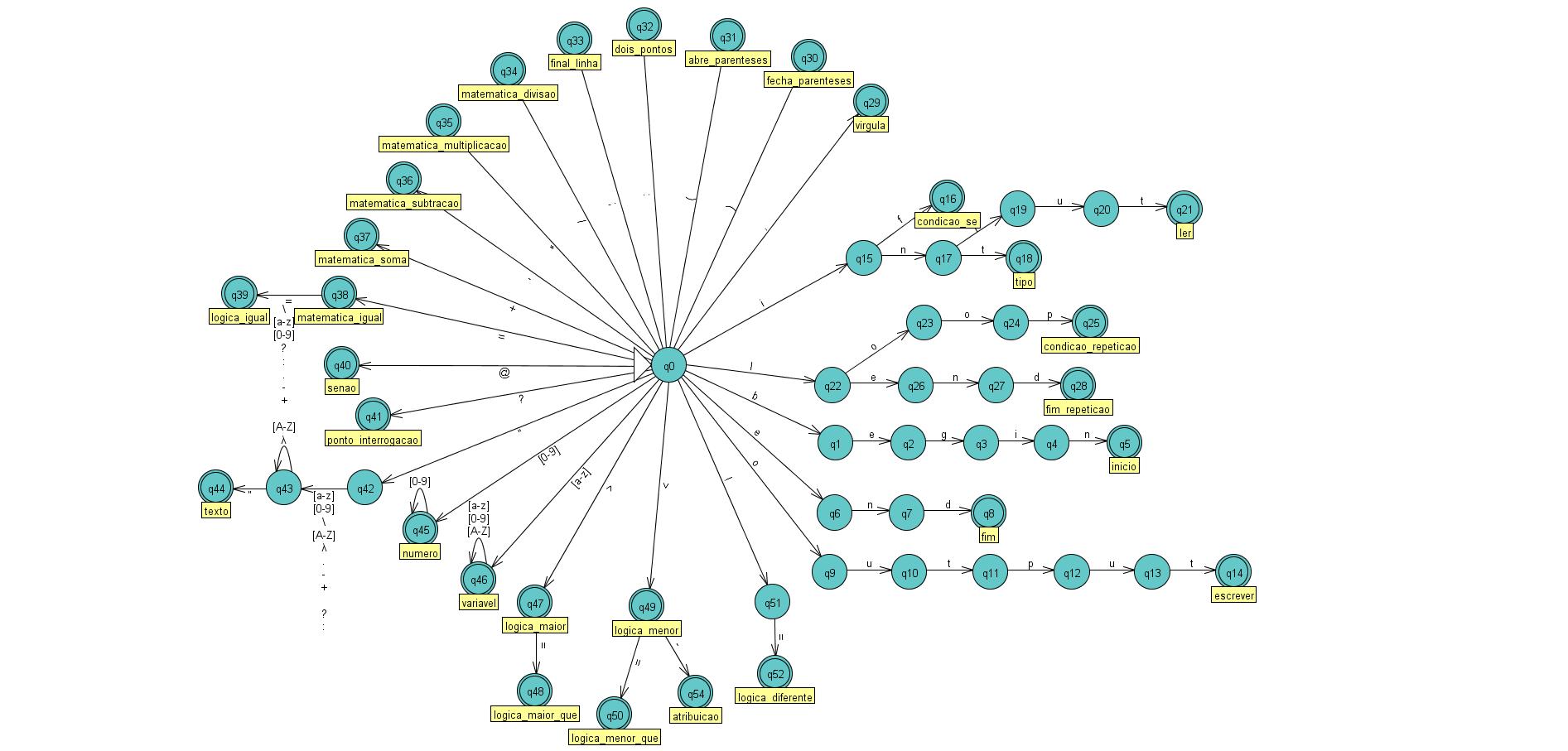
Tabela 2– Tabela de produções

|  |  |
| --- | --- |
| Não terminal | Produção |
| <INICIO\_FIM> | inicio <CODIGO> fim |
| <CODIGO> | <COMANDO> <CODIGO> |
| <CODIGO> | î |
| <COMANDO> | escrever abre\_parenteses <ARGUMENTO> fecha\_parenteses final\_linha |
| <COMANDO> | ler abre\_parenteses variavel fecha\_parenteses final\_linha |
| <COMANDO> | condicao\_se <EXPRESSAO> ponto\_interrogacao <CODIGO> <SENAO> |
| <COMANDO> | condicao\_repeticao <LOGICA> <LOOP> |
| <COMANDO> | <DECLARACAO> final\_linha |
| <COMANDO> | <ATRIBUICAO> final\_linha |
| <SENAO> | senao <CODIGO> ponto\_interrogacao final\_linha |
| <SENAO> | ponto\_interrogacao final\_linha |
| <LOOP> | abre\_parenteses <ARGUMENTO\_NUMERICO> virgula <OP\_MATEMATICA> <ARGUMENTO\_NUMERICO> fecha\_parenteses dois\_pontos <CODIGO> fim\_repeticao final\_linha |
| <LOOP> | dois\_pontos <CODIGO> fim\_repeticao final\_linha |
| <EXPRESSAO> | abre\_parenteses <LOGICA> fecha\_parenteses |
| <LOGICA> | <ARGUMENTO\_NUMERICO> <OP\_LOGICA> <ARGUMENTO\_NUMERICO> |
| <CALCULO> | matematica\_subtracao <CALCULO>|<ARGUMENTO\_NUMERICO> <EXPRESSAO\_MATEMATICA> |
| <CALCULO> | abre\_parenteses <CALCULO> fecha\_parenteses <EXPRESSAO\_MATEMATICA> |
| <EXPRESSAO\_MATEMATICA> | <OP\_MATEMATICA> <CALCULO> |
| <EXPRESSAO\_MATEMATICA> | î |
| <DECLARACAO> | tipo <ATRIBUICAO> |
| <ATRIBUICAO> | variavel <ATRIBUIR\_VALOR> |
| <ATRIBUIR\_VALOR> | atribuicao <CALCULO> |
| <ATRIBUIR\_VALOR> | î |
| <ARGUMENTO> | <ARGUMENTO\_TEXTO> | <ARGUMENTO\_NUMERICO> | î |
| <ARGUMENTO\_TEXTO> | texto <ARGUMENTO> |
| <ARGUMENTO\_NUMERICO> | numero |
| <ARGUMENTO\_NUMERICO> | variavel <ARGUMENTO |
| <OP\_LOGICA> | logica\_maior\_que |
| <OP\_LOGICA> | logica\_menor\_que |
| <OP\_LOGICA> | logica\_igual |
| <OP\_LOGICA> | logica\_maior |
| <OP\_LOGICA> | logica\_menor |
| <OP\_LOGICA> | logica\_diferente |
| <OP\_MATEMATICA> | matematica\_soma |
| <OP\_MATEMATICA> | matematica\_subtracao |
| <OP\_MATEMATICA> | matematica\_multiplicacao |
| <OP\_MATEMATICA> | matematica\_divisa |
| <OP\_MATEMATICA> | matematica\_igual |

Fonte: Elaborado pelo Autor

## Autômato

Figura 2- Autômato



Fonte: Elaborado pelo Autor

## Fases do Compilador

O Compilador desenvolvido é dividido em cinco partes:

1. Analisador Léxico: Recebe o codigo fonte, identicifa cada *token* e o salva em uma tabela de *tokens.*
2. Analisador Sintático: Recebe a tabela de *tokens*, e com ela tenta atingeir as expressões terminas da linguagem.
3. Analisador Semântico: Verifica se todas a variaveis usadas estão declaradas e seus tipos, e verifica se há opecação matemática inválida.
4. Gerador de Código Intermediário: Tem como objetivo deixa a estrutura da linguagem de forma que se assemelha a do *Assembly.*
5. Geração de Código Final: Recebe o código intermediario e converte a linguaguem para o *Assembly.*
   1. **Resultados Alcançados**

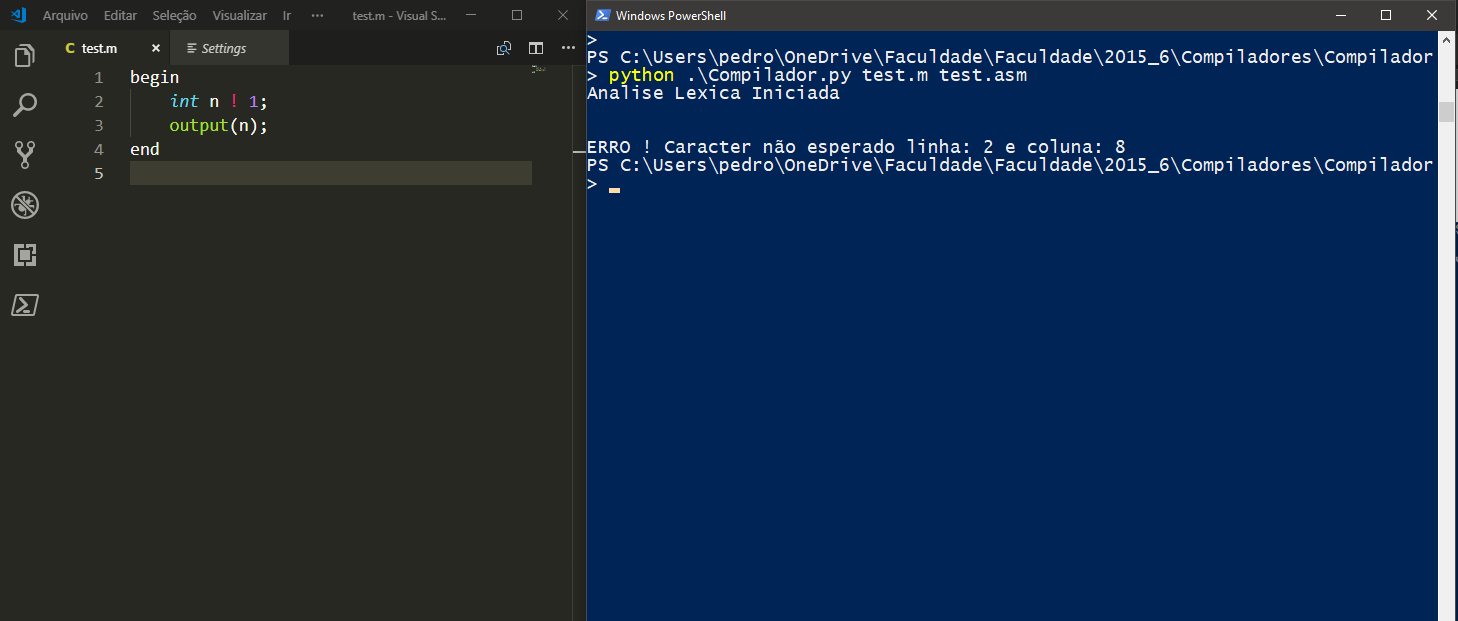
Um Compilador funcional com os principais recursos, que é capaz de ler a linguagem desensolvida e a traduzir para *Assembly* para o montador NASM.

Foi totalmente desenvolvido em *Python,* que foi uma linhagem nova e com isso permitiu um grande aprendizado.

Com a criação do compilador permitiu o aprendizado sobre o funcionamentos das linguagems de programação, permintindo assim a compreenção de toda a importacia da arquitetura do comuputador para o desenvolvimento da linguaguem.

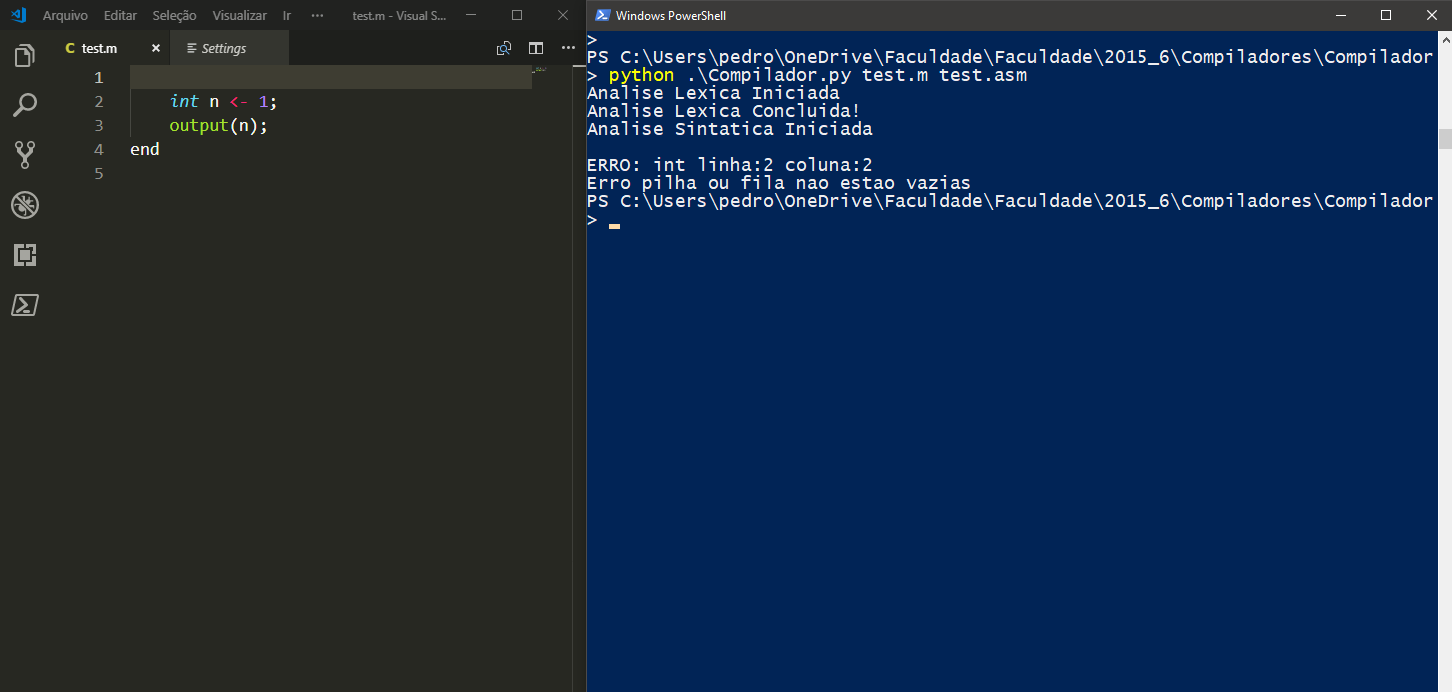
## Exemplos de Uso

Figura 3– Erro léxico



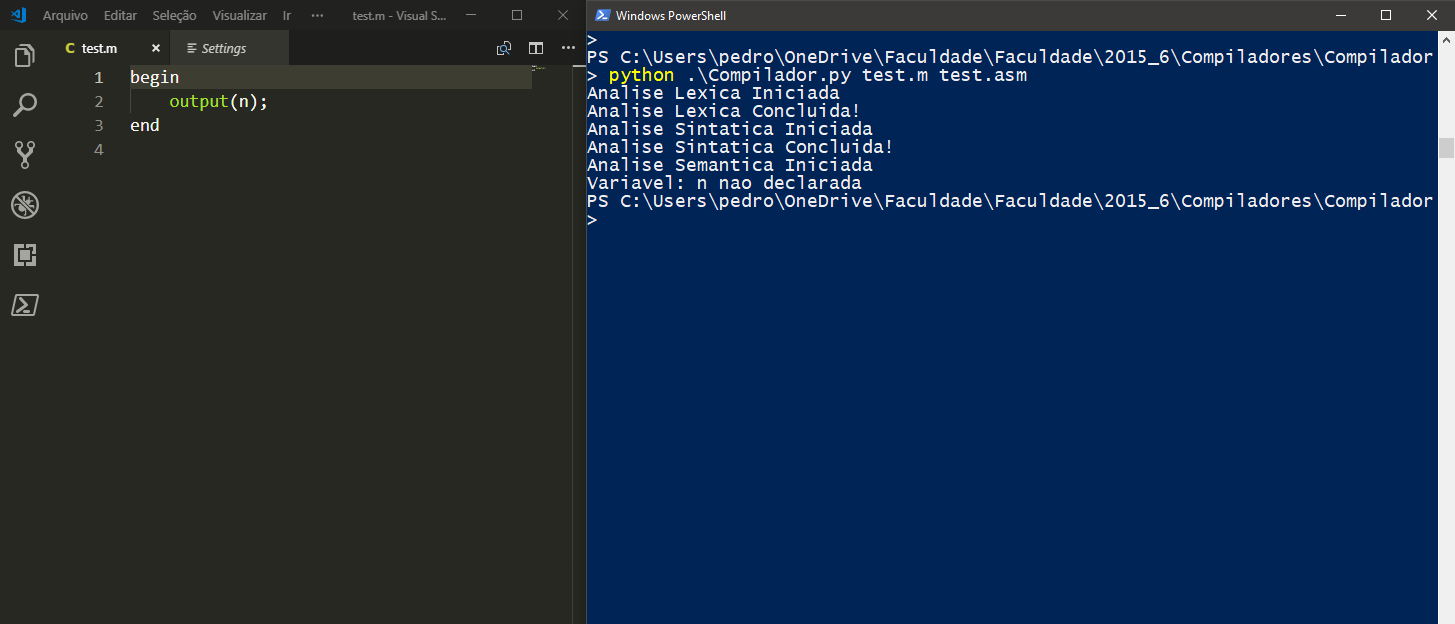
Fonte: Elaborado pelo Autor

Figura 4– Erro sintático



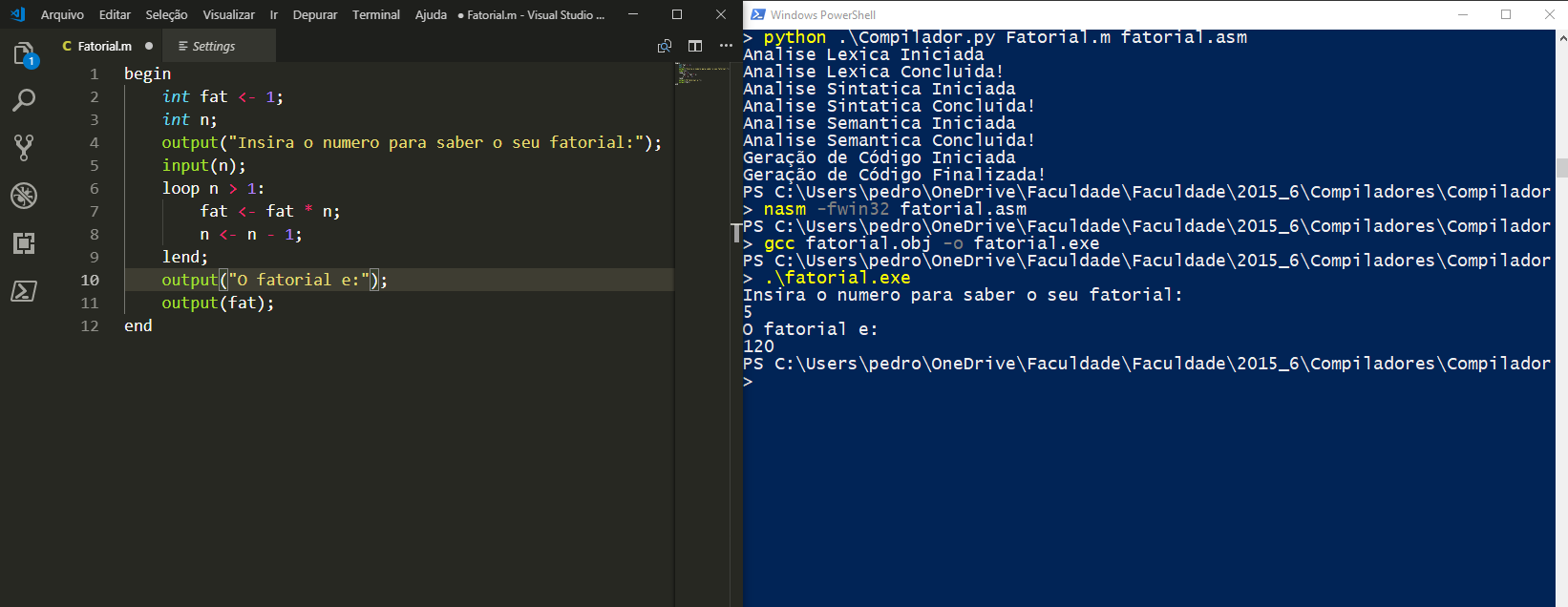
Fonte: Elaborado pelo Autor

Figura 5– Erro Semântico



Fonte: Elaborado pelo Autor

Figura 6– Fitorial de um número



Fonte: Elaborado pelo Autor

# CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O desenvolvimento do Compilador foi o maior projeto desenvolvido no Curso de Engenharia da Computação, havendo a necessidade de utilizar todo o conhecimento adquirido das matérias anteriores, como Estrutura de Dados e Arquitetura de Computadores. Assim podendo ver a importância das matérias isoladas para um grande projeto final.

Contudo, o resultado final foi satisfatório por cumprir o que foi proposto na discplina de compiladores, desde todo o conhecimento aquirido pelo o desenvolvimento até o resultado final.

# REFERÊNCIAS

AHO, Alfred V. Compiladores: Princípios, técnicas e ferramentas. Pearson. 2008.

ROGER, Susan H. *JFLAP*. Duke University, 2009. Disponível em: <http://www.jflap.org>. Acesso em: 17 dez. 2018.

*Python documentation.* Disponível em :< https://docs.python.org/3.6/# >. Acessado: em 14 dez.2018.

JETBRAINS. PyCharm. <https://www.jetbrains.com/pycharm/>. Acessado em: 17 dez. 2018.

NASM – The Netwide Assembler. < https://www.nasm.us/>. Acessado em: 17 dez. 2018