

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

<i>Figura 1 - Compilador</i>	10
<i>Figura 2- Autômato</i>	15
<i>Figura 3- Erro léxico</i>	16
<i>Figura 4- Erro sintático</i>	17
<i>Figura 5- Erro Semântico</i>	17
<i>Figura 6- Fitorial de um número</i>	18

LISTA DE TABELAS

<i>Tabela 1– Tabelade tokens</i>	12
<i>Tabela 2– Tabela de produções</i>	13

LISTA DE SIGLAS

GALS – Gerador de Analisadores Léxicos e Sintáticos

NASM – *Netwide Assembler*

SUMÁRIO

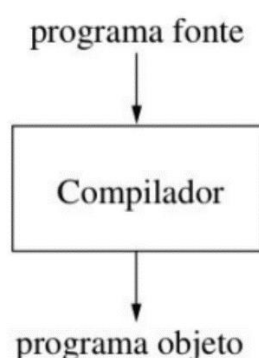
INTRODUÇÃO	10
1. Etapas do PROJETO.....	11
1.1. Metodologia	11
1.2. Objetivo Geral.....	11
1.3. Objetivos Específicos.....	11
1.4. Gramática	12
1.5. Autômato.....	15
1.6. Fases do Compilador.....	15
1.8. Exemplos de Uso	16
CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	19
REFERÊNCIAS.....	20

INTRODUÇÃO

O mundo como conhecemos depende das linguagens de programação, pois muitas pessoas usam os computadores no dia a dia, e com isso usam vários programas que foram escritos em alguma linguagem de programação. Mas antes que possa ser executados, um programa deve ser traduzido para um formato que permita ser executado por um computador. O programa que faz a tradução é conhecido como compilador.(AHO, 2008).

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

Figura 1 - Compilador



Fonte: AHO,2008

Com o avanço da tecnologia ao passar do tempo, os computadores ficaram mais acessíveis e necessários. Assim havendo cada vez a necessidade de melhoramento para que se conseguisse 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 acompanhar e melhorar o desenvolvimento das linguagens de programação e o surgimento de novas. Assim passando a gerar código para diversas plataformas diferentes.

Diante disto, esse relatório descreve o desenvolvimento de um compilador passo a passo, com todas as funções básicas necessárias para o funcionamento do mesmo, desenvolvido pelo autor na disciplina de compiladores do curso de Engenharia da Computação.

1. ETAPAS DO PROJETO

1.1. Metodologia

A primeira etapa foi desenvolvimento do compilador, foi elaborar a gramática de uma linguagem de programação, utilizando-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 utilização do *software* JFLAP (ROGER, 2009), para auxiliar no conhecimento da sintaxe da linguagem.

Na terceira etapa foi implementado o compilador, utilizando – 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* compatível com o montador NASM.

1.2. 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.
- Construir um compilar capaz de traduzir a linguagem desenvolvida para a linguagem *Assembly* para o NASM no sistema operacional *Windows*.

1.3. 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 suas etapas:
 - Analise léxica;
 - Analise Sintática;
 - Analise Semântica;
 - Geração de Código.
- Executar algoritmos na gramática desenvolvida para serem traduzidas pelo compilador.

1.4. Gramática

- A linguagem foi desenvolvida de forma que atenda os seguintes requisitos:
 - Declarar pelo menos um tipo de variável;
 - Possuir comandos de entrada e saída de dados;
 - Possuir controle de fluxo ou condicional (Se... Então...);
 - Possuir ao menos uma estrutura de repetição;
 - Reconhecer e calcular expressões matemáticas com as quatro operações básicas (adição, subtração, multiplicação e divisão);
 - Realizar as seis operações lógicas (maior igual que, maior que, menor igual que, menor que, igual e diferente);
- Baseando nos requisitos, foi desenvolvida uma gramática que:
 - Possui 29 *tokens*, 18 símbolos não terminais, 30 expressões;
 - Realiza operações numéricas inteiras;
 - Reconheça texto alfanumérico, 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 para baixo, da esquerda para a direita).

Tabela 1 – Tabela de 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

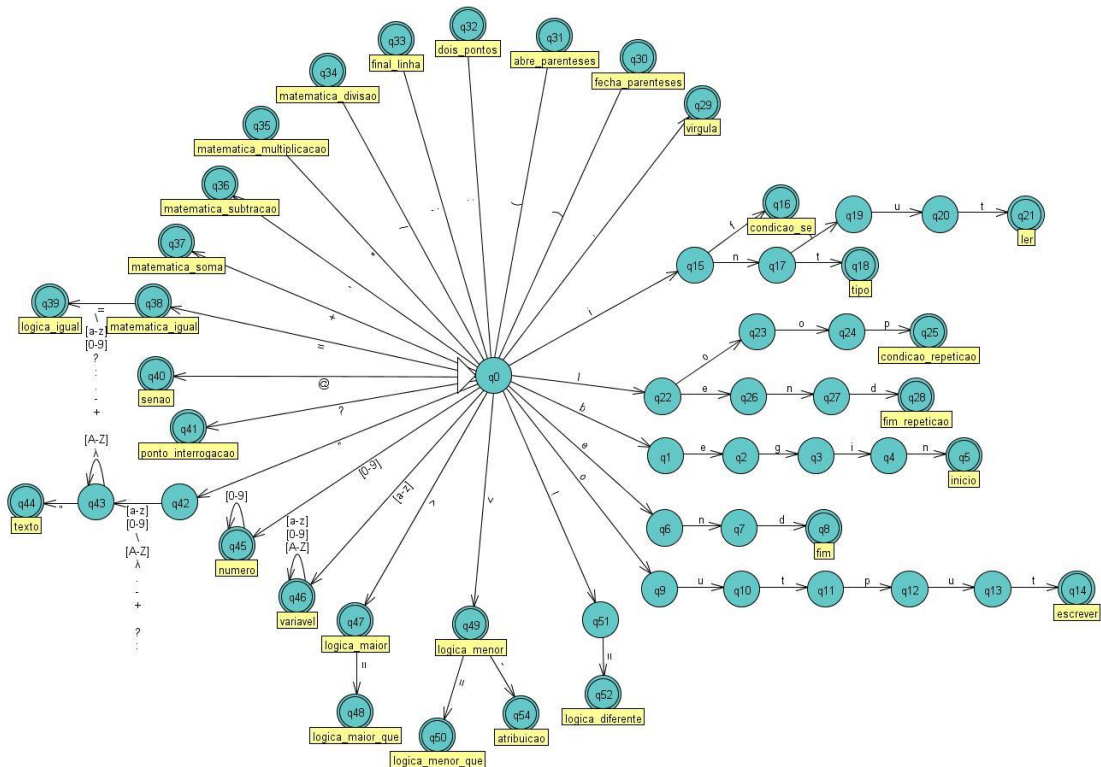
Não terminal	Produção
<INICIO_FIM>	inicio <CODIGO> fim
<CODIGO>	<COMANDO> <CODIGO>
<CODIGO>	î
<COMANDO>	escrever abre_parenteses <ARGUMENTO> fecha_parenteses final_linha
<COMANDO>	ler abre_parenteses variavel fecha_parenteses final_linha
<COMANDO>	condicao_se <EXPRESSAO> ponto_interrogacao <CODIGO> <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_divisao
<OP_MATEMATICA>	matematica_igual

Fonte: Elaborado pelo Autor

1.5. Autômato

Figura 2- Autômato



Fonte: Elaborado pelo Autor

1.6. Fases do Compilador

O Compilador desenvolvido é dividido em cinco partes:

1. Analisador Léxico: Recebe o código fonte, identifica cada *token* e o salva em uma tabela de *tokens*.
2. Analisador Sintático: Recebe a tabela de *tokens*, e com ela tenta atingir as expressões terminais da linguagem.
3. Analisador Semântico: Verifica se todas as variáveis usadas estão declaradas e seus tipos, e verifica se há operação matemática inválida.

4. Gerador de Código Intermediário: Tem como objetivo deixar a estrutura da linguagem de forma que se assemelha a do *Assembly*.
5. Geração de Código Final: Recebe o código intermediário e converte a linguagem para o *Assembly*.

1.7. Resultados Alcançados

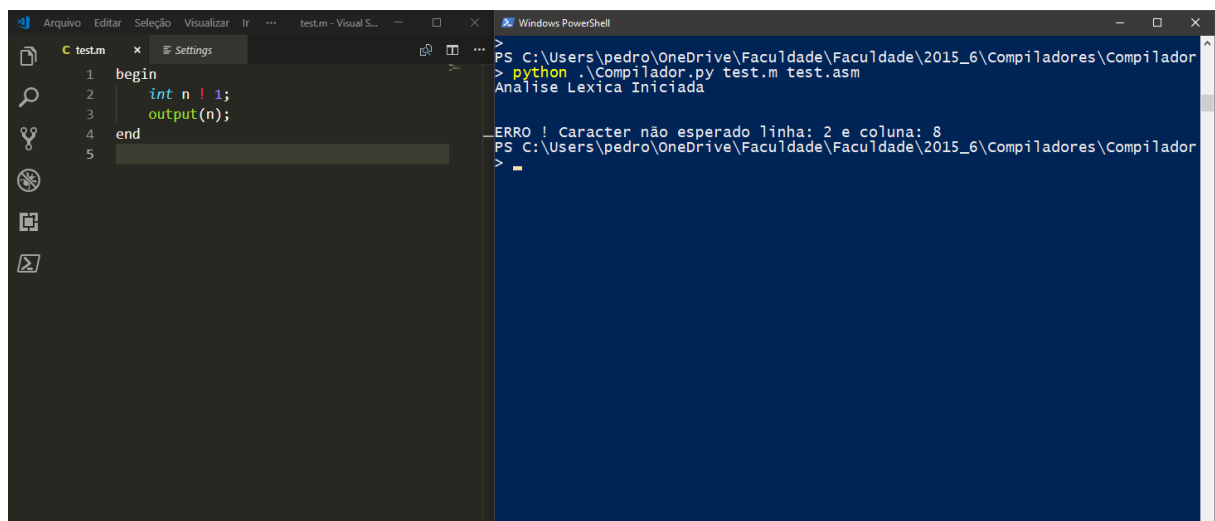
Um compilador funcional com os principais recursos, que é capaz de ler a linguagem desenhada e a traduzir para *Assembly* para o montador NASM.

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

Com a criação do compilador permitiu o aprendizado sobre o funcionamento das linguagens de programação, permitindo assim a compreensão de toda a importância da arquitetura do computador para o desenvolvimento da linguagem.

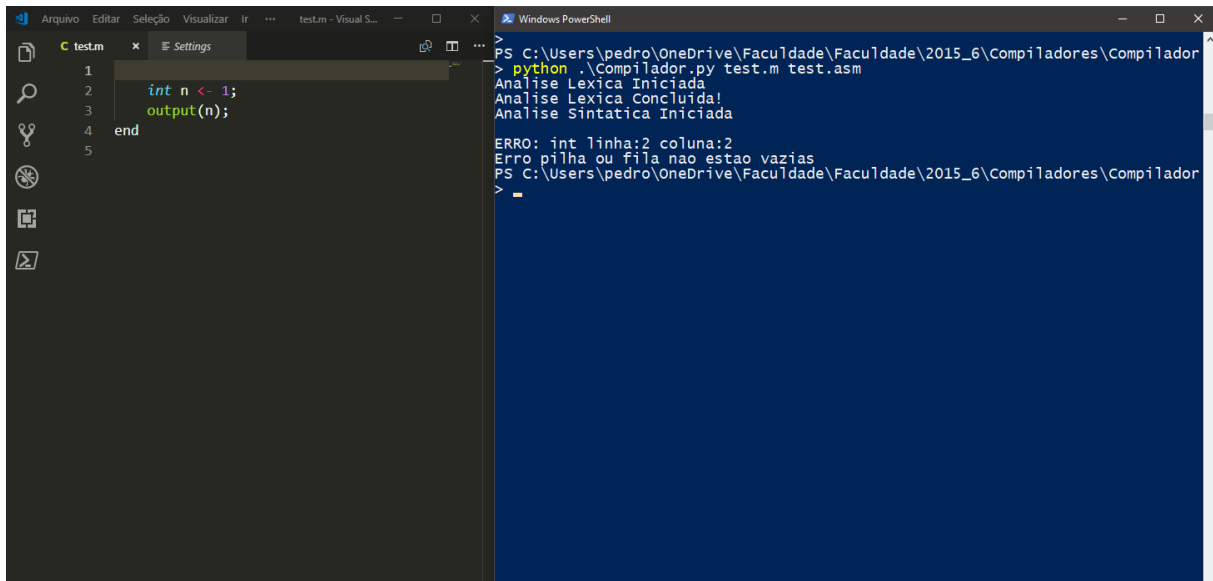
1.8. Exemplos de Uso

Figura 3– Erro léxico



Fonte: Elaborado pelo Autor

Figura 4– Erro sintático



The screenshot shows a code editor on the left with a file named 'test.m'. The code contains the following lines:

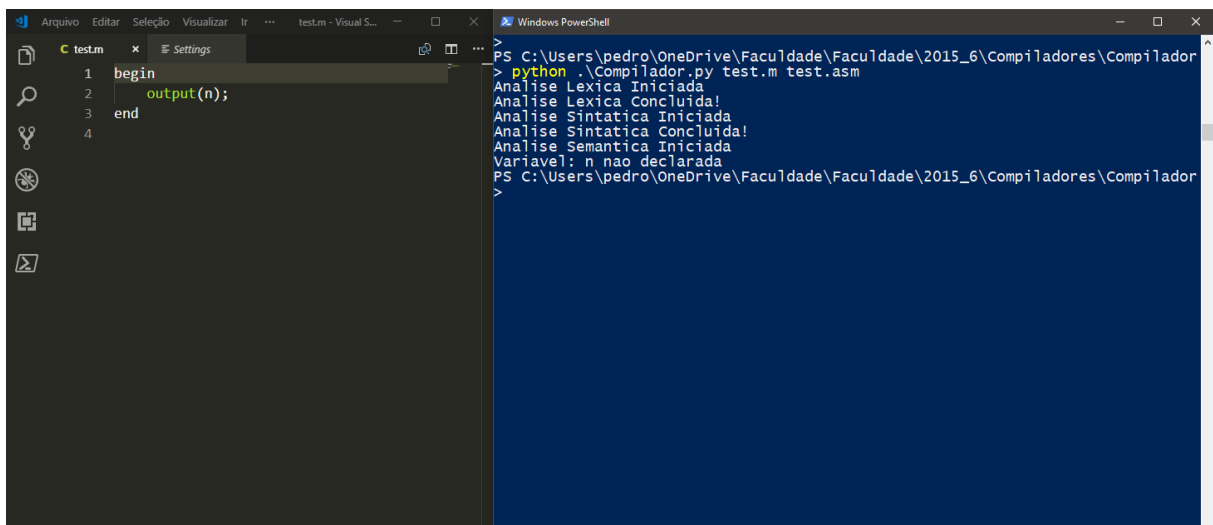
```
1  
2  int n <- 1;  
3  output(n);  
4  
5  end
```

On the right, a Windows PowerShell terminal window displays the output of a Python script. The script runs successfully through lexical and syntactic analysis, but then fails during semantic analysis with the following error message:

```
> PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador  
> python .\Compilador.py test.m test.asm  
Analise Lexica Iniciada  
Analise Lexica Concluida!  
Analise Sintatica Iniciada  
ERRO: int linha:2 coluna:2  
Erro pilha ou fila nao estao vazias  
> PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador  
>
```

Fonte: Elaborado pelo Autor

Figura 5– Erro Semântico



The screenshot shows a code editor on the left with a file named 'test.m'. The code contains the following lines:

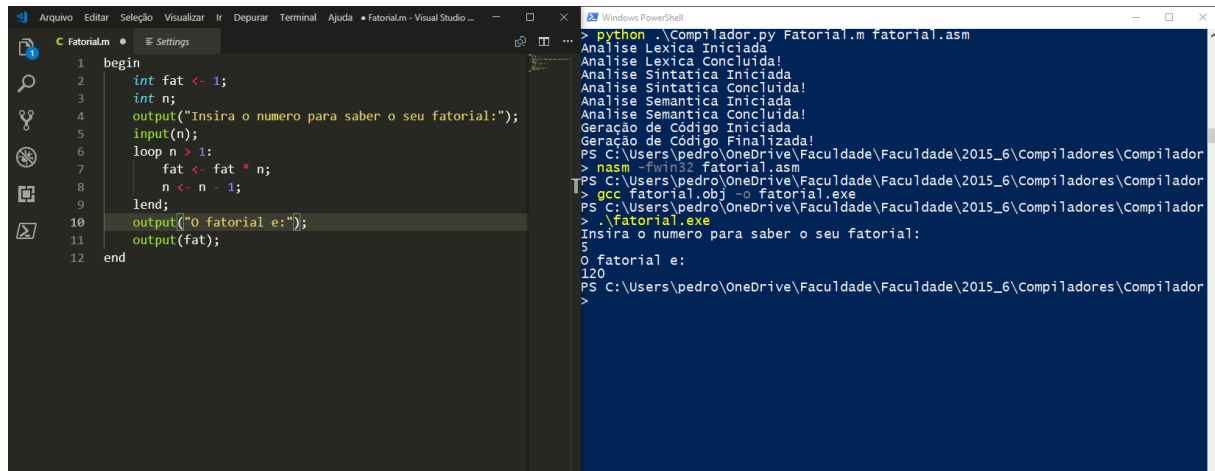
```
1  begin  
2  output(n);  
3  
4  end
```

On the right, a Windows PowerShell terminal window displays the output of a Python script. The script runs successfully through lexical and syntactic analysis, but then fails during semantic analysis with the following error message:

```
> PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador  
> python .\Compilador.py test.m test.asm  
Analise Lexica Iniciada  
Analise Lexica Concluida!  
Analise Sintatica Iniciada  
Analise Sintatica Concluida!  
Analise Semantica Iniciada  
Variavel: n nao declarada  
> PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador  
>
```

Fonte: Elaborado pelo Autor

Figura 6– Fatorial de um número



The image shows a screenshot of a development environment with two windows. The left window is Visual Studio, displaying a Pascal-like source file named 'Fatorial.m'. The code defines a function 'fatorial' that calculates the factorial of a number 'n' using a loop. The right window is a Windows PowerShell terminal, showing the compilation process. It starts with running a Python compiler script, which performs lexical, syntactic, and semantic analysis, and generates assembly code. Then, it uses NASM to assemble the code into an object file, and GCC to link it into an executable named 'fatorial.exe'. Finally, it runs the executable, which prompts for an input number and outputs the result, 120.

```
1 begin
2   int fat <- 1;
3   int n;
4   output("Insira o numero para saber o seu fatorial:");
5   input(n);
6   loop n > 1:
7     fat <- fat * n;
8     n <- n - 1;
9   lend;
10  output("O fatorial e:");
11  output(fat);
12 end
```

```
> python .\Compilador.py Fatorial.m fatorial.asm
Analise Lexica Iniciada
Analise Lexica Concluida!
Analise Sintatica Iniciada
Analise Sintatica Concluida!
Analise Semantica Iniciada
Analise Semantica Concluida!
Geração de Código Iniciada
Geração de Código Finalizada!
PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador> nasm -fwin32 fatorial.asm
PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador> gcc fatorial.obj -o fatorial.exe
PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador> .\fatorial.exe
Insira o numero para saber o seu fatorial:
5
O fatorial e:
120
PS C:\Users\pedro\OneDrive\Faculdade\Faculdade\2015_6\Compiladores\Compilador>
```

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 disciplina de compiladores, desde todo o conhecimento adquirido 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