

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANTONIELLI DALCIM DO NASCIMENTO NUNES
OBENSON MAURICE

CRIAÇÃO DE UM COMPILADOR BÁSICO

CHAPECÓ
2023

RESUMO

Este artigo tem por finalidade a apresentação das etapas e os fundamentos necessários para a criação de um compilador, como a Análise Léxica, Análise Sintática, Análise Semântica, Geração de código intermediário.

Neste artigo será apresentado o desenvolvimento e os resultados obtidos ao implementar um compilador e seus processos de compilador.

Palavras-chave: Compilador. Etapas. Análise. Léxica. Sintática. Semântica. Código Intermediário.

1 – INTRODUÇÃO

Um compilador é um programa que recebe um programa escrito em uma linguagem de programação (linguagem fonte) como entrada e produz a saída de um programa escrito em outra linguagem (linguagem objeto) a partir de seu processamento. Se a linguagem fonte for uma linguagem de alto nível como Python ou Java, e a linguagem objeto for uma linguagem de baixo nível, como linguagem assembly ou linguagem de máquina, o tradutor é chamado de compilador.

Hoje, mais de 50 anos após o conceito que levou ao desenvolvimento dos compiladores, essa situação ainda está longe do ideal. Existem muitos tipos de linguagens de programação e ainda existem especialistas em uma linguagem ou outra. Mas o certo é que, se não houver compilador, o desenvolvimento da programação será mais primitivo que o estágio atual. O compilador traz código-fonte escrito em linguagem de alto nível, em um código semanticamente equivalente, escrito em código de máquina.

Nesse artigo será utilizado uma ferramenta externa chamada Gold Parser para a criação da tabela LALR, utilizada para a verificação sintática, está será aplicada na linguagem base, a fim de economizar esforço, tempo e evitar error.

2 – PRINCÍPIOS DE UM COMPILADOR

Um compilador é um software que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Em geral, um compilador não produz diretamente o código de máquina, mas sim, um programa em linguagem simbólica (assembly) semanticamente equivalente ao programa em linguagem de alto nível. O programa em linguagem simbólica é, então, traduzido para o programa em linguagem de máquina através de montadores.

Para realizar esta tarefa, o compilador realiza uma análise lexical, sintática e semântica do código-fonte do programa executado em uma linguagem abstrata e, a seguir, gera o código de máquina. Com o advento dos computadores de programas armazenados de John Von Neumann no final dos anos 1940, tornou-se necessário escrever sequências de código (ou Programa) para que o computador possa realizar os cálculos necessários. Inicialmente, esses programas eram escritos em linguagem de máquina, ou seja, códigos digitais que representam as operações da máquina e podem ser executados de forma eficiente (LOUDEN, 2004). Ao longo da década de 1950, ficou conhecido que compiladores são programas difíceis de escrever. Por exemplo, o primeiro compilador FORTRAN levou 18 homem-anos para programar, o que levou à descobertas técnicas sistemáticas para o tratamento de muitas das mais importantes tarefas que ocorreram durante a compilação, e seguindo esta evolução, as técnicas de implementação em linguagens abstratas também se tornaram cada vez mais avançadas (AHO, 1996).

O compilador tem duas tarefas básicas: Análise, que examina, verifica e compreende o texto de entrada (no código-fonte); Geração de código, em que o texto de saída (código de máquina) é gerado de forma correspondente ao texto de entrada (código-fonte da linguagem abstrata).

2.1 – ANALISADORES: LÉXICO, SINTÁTICO E SEMÂNTICO

A análise lexical é o processo de analisar a entrada de linhas de caracteres (como o código-fonte de um programa de computador) e gerar uma série de símbolos chamados "símbolos lexicais" (lexical tokens) ou simplesmente "símbolos" (tokens), que podem ser manipulados mais facilmente por um parser (leitor de saída). A análise lexical é um método de verificação de um determinado alfabeto. Quando analisamos uma palavra, podemos usar a análise lexical para definir se há caracteres que não pertencem ao nosso alfabeto, ou ao alfabeto que inventamos. O analisador léxico é a primeira etapa do compilador, depois é a etapa da análise sintática (LOUDEN, 2004). O analisador sintático é considerado o núcleo do compilador e é responsável por verificar se a sequência de símbolos contida no código-fonte constitui um programa válido. A gramática de uma linguagem de programação pode ser descrita por uma gramática livre de contexto.

O analisador sintático recebe um conjunto de tokens do analisador léxico e usa um conjunto de regras para construir uma árvore gramatical estruturada. A maioria dos analisadores implementados em compiladores aceita algum tipo de linguagem livre de contexto para análise (LOUDEN, 2004).

A análise semântica é o terceiro estágio da compilação, e seu nome é assim porque ela precisa calcular informações além do escopo da gramática livre de contexto e dos algoritmos de análise sintática padrão, portanto, essas informações não podem ser consideradas gramática. As informações calculadas também estão intimamente relacionadas ao significado do programa de

tradução, ou seja, semântica.(LOUDEN, 2004). É neste analisador que se verifica o erro semântico no código-fonte, ou seja, se não houver contradição com o significado da estrutura utilizada pelo programador, ele coletará as informações necessárias para a próxima etapa de compilação, e assim é a geração do código fonte.

2.2 – GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

Certos tipos de tradutores convertem linguagens de programação em linguagens simplificadas, chamadas de códigos intermediários, que podem ser executados diretamente por programas chamados de intérpretes. Pode-se pensar no código intermediário como uma linguagem de máquina de computador abstrata projetada para executar o código-fonte.

No estágio intermediário de geração de código, a árvore de sintaxe é transformada em uma representação intermediária do código-fonte. Essa linguagem intermediária está mais próxima da linguagem objeto do que do código-fonte, mas ainda permite uma operação mais fácil do que usar o código assembly ou código de máquina. Em um programa de tradução muito simples, a geração do código intermediário pode ser estruturalmente distribuída nas etapas anteriores (análise sintática e análise semântica), ou mesmo inexistente (convertida diretamente em código-alvo).

3 – IMPLEMENTAÇÃO E RESULTADOS

O objetivo de criar um compilador é resolver problemas específicos por meio de uma nova linguagem abstrata (linguagem de alto nível), determinar a finalidade da linguagem e entender se ela será usada para fins comerciais, científicos, entre outros.

Com isso, foi construído o código com base na linguagem de programação python, devido sua facilidade de manipular estruturas.

O código foi implementado esperando atender as exigências vistas em aula. Para isso foi construído um autômato finito determinístico. Após, foi então criado um arquivo que representa o código (pseudo-código) escrito pelo programador, a fim de validação. Foi criada a gramática livre de contexto utilizada para definir as estruturas de linguagem.

Tem-se como arquivos de entrada do programa o arquivo de tokens, que é composto pela GLC e tokens utilizados, um arquivo para armazenar o pseudo-código escrito pelo programador que será analisado, o arquivo de linguagem bnf, responsável pelo controle de estruturas permitidas na linguagem. E por último, um arquivo xml, contendo as informações para a comparação sintática, gerado pelo programa Gold Parser.

O arquivo de entrada contendo o pseudo-código escrito pelo programador será analisado e se não ocorrer nenhum erro, ele será compilado. Caso algum erro ocorra, o erro será apresentado. Os possíveis erros são:

A análise léxica varre todos os caracteres do arquivo de entrada, verificando se eles fazem parte da linguagem, e realizando uma determinada ação a cada caractere encontrado, adicionando-o a fita de saída e na tabela de símbolos.

Essa análise leva como base a ocorrência ou não na linguagem hipotética. A linguagem hipotética apresenta 4 ações possíveis:

- Atribuição de um valor a uma variável,
- Declaração da variável,
- Verificação de condição
- Uma estrutura de loop.

Sintática: recebe como entrada, o resultado da análise léxica. Procura por erros na estrutura sintática do programa, verificando se obedece às regras da gramática livre de contexto.

Semântica: Como semântica, definimos que seriam tratados os casos em que:

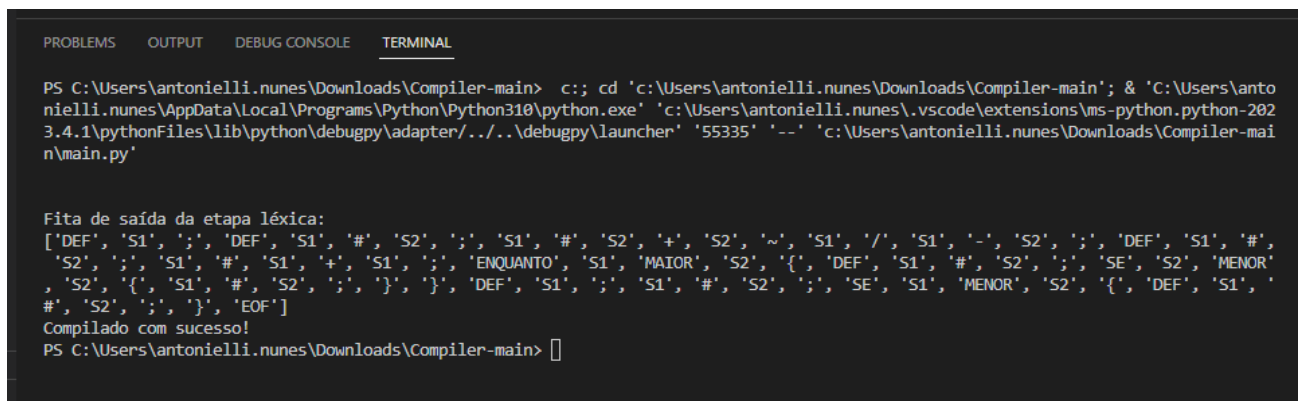
- Uma variável já foi declarada anteriormente;
- Uma variável está sendo usada fora do escopo em que foi declarada;
- Uma variável utilizada não foi declarada;

Com relação à geração de código intermediário, não foi possível concluir a produção completa.

Utilizando os conhecimentos adquiridos em aula, foi possível alcançar resultados válidos, com margem de funcionamento satisfatória. Situações como escopos vazios (representados por {}) não foram trabalhadas, e podem retornar erros sintáticos, já que o programa reconhece '}' como sendo um carácter, não um operando, porém, dentro de uma margem tolerável.

Pode-se dizer portanto que o trabalho foi gerado satisfatoriamente.

Implementado a leitura e tratamento de erros do Analisador Léxico, Analisador Sintático e Analisador Semântico e Código Intermediário

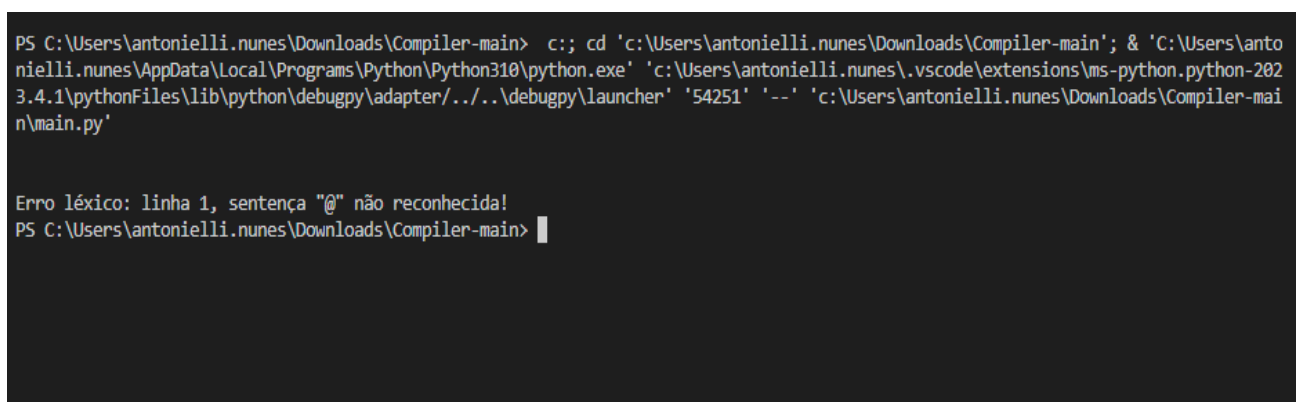


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55335' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Fita de saída da etapa léxica:
['DEF', 'S1', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S2', '+', 'S2', '~', 'S1', '/', 'S1', '-', 'S2', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S1', '+', 'S1', ';', 'ENQUANTO', 'S1', 'MAIOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', 'SE', 'S2', 'MENOR', 'S2', '{', 'S1', '#', 'S2', ';', '}', '}', 'DEF', 'S1', ';', 'S1', '#', 'S2', ';', 'SE', 'S1', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', '}', '}', 'EOF']
Compilado com sucesso!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

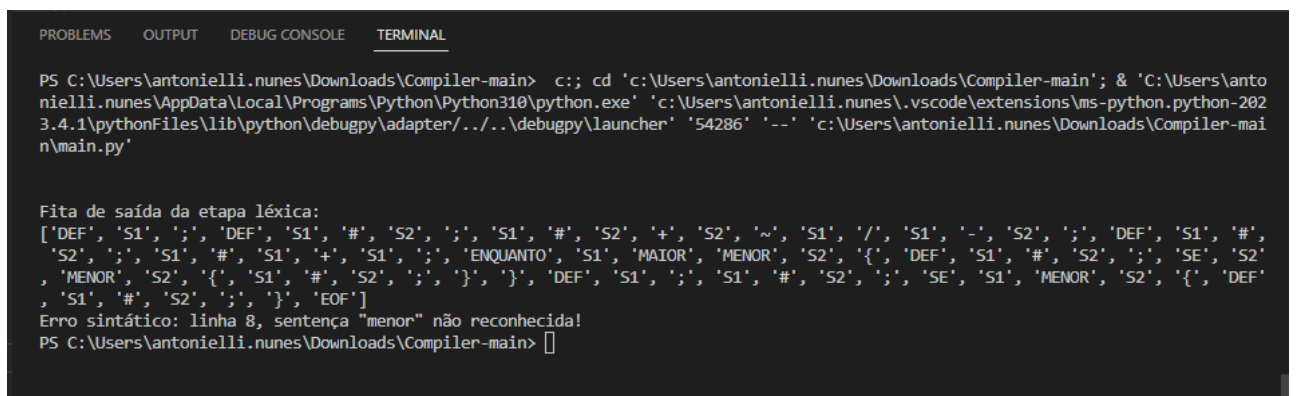
Figura 1 – Compilado com sucesso



```
PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54251' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Erro léxico: linha 1, sentença "@" não reconhecida!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

Figura 2 – Análise Léxica tratamento de erro



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54286' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Fita de saída da etapa léxica:
['DEF', 'S1', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S2', '+', 'S2', '~', 'S1', '/', 'S1', '-', 'S2', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S1', '+', 'S1', ';', 'ENQUANTO', 'S1', 'MAIOR', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', 'SE', 'S2', 'MENOR', 'S2', '{', 'S1', '#', 'S2', ';', '}', '}', 'DEF', 'S1', ';', 'S1', '#', 'S2', ';', 'SE', 'S1', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', '}', '}', 'EOF']
Erro sintático: linha 8, sentença "menor" não reconhecida!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

Figura 3– Análise Sintática tratamento de erro

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54297' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Fita de saída da etapa léxica:
['DEF', 'S1', ';', 'DEF', 'S1', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S2', '+', 'S2', '~', 'S1', '/', 'S1', '-', 'S2', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S1', '+', 'S1', ';', 'ENQUANTO', 'S1', 'MAIOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', 'SE', 'S2', 'MENOR', 'S2', '{', 'S1', '#', 'S2', ';', '}', '}', 'DEF', 'S1', ';', 'S1', '#', 'S2', ';', 'SE', 'S1', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', '}', '}', 'EOF']
Erro semântico: linha 2, variável "a" já declarada!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

Figura 4 – Análise Semântica tratamento de erro de variável já declarada

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54304' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Fita de saída da etapa léxica:
['DEF', 'S1', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S2', '+', 'S2', '~', 'S1', '/', 'S1', '-', 'S2', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S1', '+', 'S1', ';', 'ENQUANTO', 'S1', 'MAIOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', 'SE', 'S2', 'MENOR', 'S2', '{', 'S1', '#', 'S2', ';', '}', '}', 'DEF', 'S1', ';', 'S1', '#', 'S2', ';', 'SE', 'S1', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', '}', '}', 'EOF']
Erro semântico: linha 19, variável "y" não declarada!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

Figura 5 – Análise Semântica tratamento de erro de variável não declarada

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\antonielli.nunes\Downloads\Compiler-main> c:: cd 'c:\Users\antonielli.nunes\Downloads\Compiler-main'; & 'C:\Users\antonielli.nunes\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\antonielli.nunes\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '54311' '--' 'c:\Users\antonielli.nunes\Downloads\Compiler-main\main.py'

Fita de saída da etapa léxica:
['DEF', 'S1', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S2', '+', 'S2', '~', 'S1', '/', 'S1', '-', 'S2', ';', 'DEF', 'S1', '#', 'S2', ';', 'S1', '#', 'S1', '+', 'S1', ';', 'ENQUANTO', 'S1', 'MAIOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', 'SE', 'S2', 'MENOR', 'S2', '{', 'S1', '#', 'S2', ';', '}', '}', 'DEF', 'S1', ';', 'S1', '#', 'S2', ';', 'SE', 'S1', 'MENOR', 'S2', '{', 'DEF', 'S1', '#', 'S2', ';', '}', '}', 'EOF']
Erro semântico: linha 21, variável "abcy" escopo inválido!
PS C:\Users\antonielli.nunes\Downloads\Compiler-main>
```

Figura 6 – Análise Semântica tratamento de erro de variável fora do escopo

4 – CONCLUSÕES

Percebeu-se com esse projeto que, com uma base tão técnica e científica a matéria, a matéria de compiladores se torna maçante, mas se uma pesquisa mais aprofundada for conduzida de uma forma mais útil e teórica, se torna muito útil para a construção de algoritmos mais confiáveis e estáveis.

No decorrer do projeto, constatou-se que, com a teoria relativamente colocada em prática, os assuntos tratados ficam mais fáceis de serem compreendidos.

Mesmo assim, a criação de um simples compilador foi bem difícil de ser concebido, pois suas etapas precisam de uma alta compreensão de matérias antes vista pelo curso de Ciência da Computação, como: Fundamentos de Informática, Matemática, Arquitetura de Computadores, Teoria da Computação e Linguagem Formais e Autômatos.

A estrutura geral do compilador está diretamente ligada com seus analisadores, tabelas e resultados, com determinação e pesquisa o código sempre pode ser mais bem apreendido e consequentemente melhorado.

5 – REFERÊNCIAS BIBLIOGRÁFICAS

LOUDEN, K.C. Compiladores – Princípios e Práticas. Ed. Thomson Pioneira, 2004.

AHO, S. Compiladores: Princípios, técnicas e ferramentas, Ed. LTC. 1996.