Compilador Food

Felipe Henrique de Moraes, Kaique Santos de Andrade

Universidade Tecnológica Federal do Paraná (UTFPR)

Medianeira – PR – Brazil

Resumo. *Este artigo visa explorar os princípios, técnicas e etapas fundamentais de um compilador. Abordará desde a análise léxica, sintática e semântica até a geração de código intermediário. O foco será na criação de um compilador para uma linguagem fictícia denominada Food, demonstrando a aplicação prática dos conceitos aprendidos.*

1. Introdução

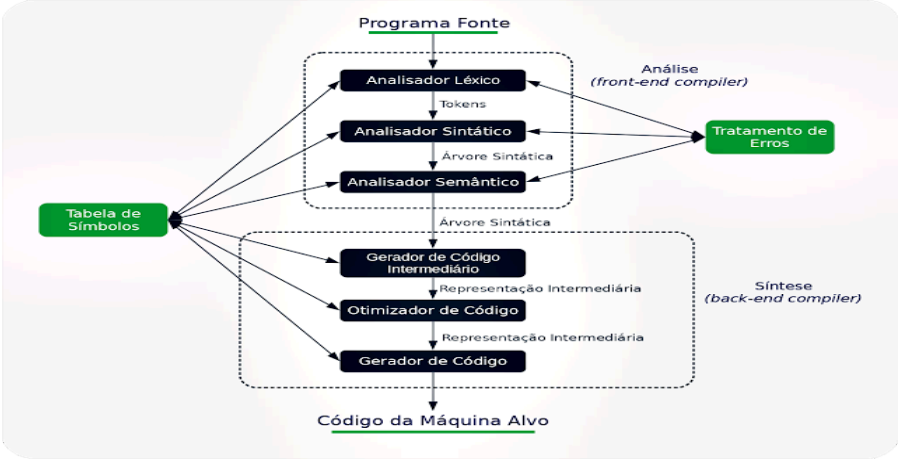
O objetivo principal nesse artigo é entender os princípios básicos de um compilador e todas as suas etapas focando na construção de uma compilador prático e simples com uma gramática regular usando conceitos de comidas da culinária brasileira para definir a sua estrutura.

2. Definição de um compilador

Um compilador é um software tradutor que lê um código-fonte escrito em uma linguagem de alto nível para linguagem de baixo nível linguagem de máquina [1]. cada compilador vai gerar o código específico para cada máquina, de acordo com seu hardware, A necessidade de um compilador surge principalmente da dificuldade em escrever código em linguagem de baixo nível, como a linguagem de máquina, devido às diversas arquiteturas de processador, cada uma com suas próprias instruções. Além disso, a interpretação direta de instruções em linguagem binária não é eficiente e torna a legibilidade, manutenção e compreensão do código mais desafiadoras.

2.1 Etapas de um compilador

Figura 1:Etapas do processo de compilação



Fonte:Disponivel em
https://www.cadcobol.com.br/compiladores_wikipedia_autociencia.png

Um detalhe pertinente sobre as análises é que elas não ocorrem em uma ordem sequencial, mas sim em paralelo e simultaneamente, de acordo com as entradas do código-fonte.

2.2 Análise Léxica (Scanner):

Um programa que realiza a análise léxica pode ser conhecido como lexer tokenizer ou scanner[2], é a primeira etapa do processo de compilação a , nessa etapa análise léxica irá fazer a remoção dos espaços em branco e comentários uma vez que são usados apenas para melhorar a visualização do código para o olho humano.

Durante análise léxica é gerada a tabela de símbolos formação de tokens que se dá pela leitura do código fonte caracter a caracter e validando se pertence a linguagem ou não e isso ocorre simultaneamente com a tabela de símbolos armazenando as informações é usada e atualizada durante várias etapas da compilação.

análise léxica fica responsável pelos erros encontrados que não são reconhecidos pela linguagem que são caracter não previstos sequências inválidas, fim de arquivo inesperado EOF durante o reconhecimento. a Invocação sempre ocorre quando um novo

token é necessário e isso gera todas as chamadas necessárias prejudicando o desempenho do processo.

Instrução	Erro
int meu&Variavel = 10;	Contém um caractere inválido "&"
char letra = 'A;	Ausência de caracteres 'A'

2.3 Análise Sintática (Parser)

O papel deste analisador é avaliar se a sequência de tokens recebida do scanner pode ser gerada através da gramática formal da linguagem alvo e construir uma árvore gramatical correspondente a esta sequência[3].

Durante a análise sintática, o compilador verifica se o código fonte está corretamente estruturado de acordo com as regras gramaticais da linguagem de programação. Ele se concentra em garantir que a sequência de tokens formada pelo código fonte siga a sintaxe definida pela gramática da linguagem.

No entanto, nesta fase, o compilador não avalia se o código fonte faz sentido em termos de lógica ou semântica do programa. Ele apenas verifica se a estrutura do código está de acordo com as regras gramaticais da linguagem. Isso significa que o compilador não considerará se o código está logicamente correto ou se produzirá os resultados desejados quando executado.

O exemplo dado sobre o uso de ponto e vírgula em linguagens como C ou Java, e a tabulação em Python, ilustra como a estrutura do código fonte deve seguir as regras gramaticais específicas de cada linguagem durante a análise sintática.

Instrução	erro
int x = 5 int y = 10;	Ausência do “;”
int y = x * * 5;	Uso incorreto do operador ”*”
printf("Hello, world!"	Ausência do “);”

2.4 Análise Semântica

A análise semântica é responsável por verificar características relacionadas ao significado das instruções. O Analisador Semântico faz uso da árvore sintática e da tabela de símbolos para realizar a análise semântica[3].De maneira genérica, podemos dizer que a análise semântica verifica se o código tem significado lógico e se suas construções estão em conformidade com as regras e expectativas da linguagem de programação.

Instrução	Erro

int x = 5;
char y = 'A';
int z = x + y;
	tentativa de somar um inteiro com um caractere

int main()
{
  int x = 5;
  printf("%d", y);
  return 0;
}
	a variável 'y' não foi declarada

2.5 back-end

O processo do back-end vai utilizar toda a estrutura fornecida e gerada pelo front-end fazendo a optimização e gerando para a arquitetura alvo do processador em específico .

2.6 Processo de Otimização

A transformação do código gerado para que reduza o consumo de recursos computacionais é essencial para qualquer arquitetura, a redução de memŕoia, recursos de rede, essa fase vem complementando as outras fases geração de código que também vai atuar simultaneamente, cada etapa do código vai ter sua etapa como local em que cada instrução é observada de forma individual ou em bloco em a sequência de instruções e globais.

3. Linguagem Food

A linguagem Food é baseada em palavras chaves na culinária em geral esse tema foi escolhido por temos muitas facilidade de identificar essas palavras e compreender o código sem muita dificuldade onde cada espectador vai poder ter uma noção básica o que são palavras da linguagem ou não. Foi desenvolvida como gramática regular possui também variáveis tipadas onde deve ser informada qual tipo de cada alocação para o programa a sua estrutura padrão é baseada em C com nomeação de blocos definição de variáveis abertura e fechamento de blocos;

Figura 4. Food Gramática.

4.1.1 tipos de dados

```
<TPSTRING>  ::= torrada
<TPREAL>    ::= agua
<TIPOBOOL>  ::= processado
<TPREAL>    ::= agua
```

4.1.2 Estrutura básica do programa

```
<INICIOPROGRAMA>::= prato
<ABREBLOCO>     ::= {
<FECHABLOCO>    ::= }
<ABREBLOCOND>   ::= (
<FECHABLOCOND>  ::= )
< FIM_STRUC>    ::= .
```

4.1.3 Estrutura condicional

```
<CONDIF>        ::= almoco
<CONDELSE>      ::= sobremesa
<REPWHILE>      ::= rodizio
<REPFOR>        ::= degustacao
<ENTAO>         ::= coma
```

4.1.4 Tokens IDs

```
< NUMERO>       ::= (< DIGITO >)+
< #DIGITO>      ::= [ 0-9 ]
< ID>           ::= <LETRA> (< LETRA> | < DIGITO>)*
< LETRA>        ::= [A-Z] | [a-z]
< BOOL>         ::= (TRUE | FALSE | true | false)
< ASPAS>        ::= '
```

4.1.5 Operadores lógicos e matemáticos

```
<MENORIGUAL> ::=    <=
<MAIORIGUAL>::=     >=
<IGUAL>     ::=     ==
<DIFERENTE> ::=     !=
<E>         ::=     &&
<OU>        ::=     ||
<MAIS>      ::=     +
<MENOS>     ::=     -
<MULTI>     ::=     *
<DIV>       ::=     /
<MAIOR>     ::=     >
```

```
<MENOR>     ::=     <
<ATRIBUI>   ::=     =
```

4.1.6 operação de comparação

```
<MAIOR> | <MENOR> | <MENORIGUAL> | <MAIORIGUAL> | <DIFERENTE> | <IGUAL
```

4.1.7 Tipagem de dados

```
<TIPOINTEIRO > | <TIPOBOOL > | <TPSTRING > | <TPREAL>
```

4.1.8 Declaração de variáveis

```
  <TIPOINTEIRO> (<ID> (atribuicaoVar())* <FIM_STRUC>)+
  | <TIPOBOOL> (<ID> (atribuicaoVar())* <FIM_STRUC>)+
  | <TPSTRING> (<ID> (atribuicaoVar())* <FIM_STRUC>)+
```

4.1.9 estrutura principal

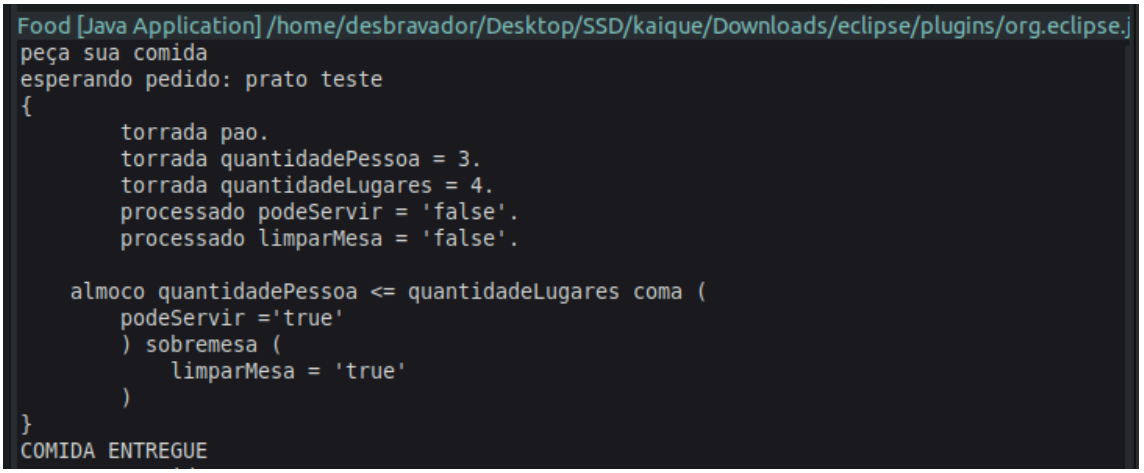
```
  <INICIOPROGRAMA> <ID>
  <ABREBLOCO>
      <declara_variavel>
      <body>
  <FECHABLOCO>
```

4.2.1 Exemplo executável

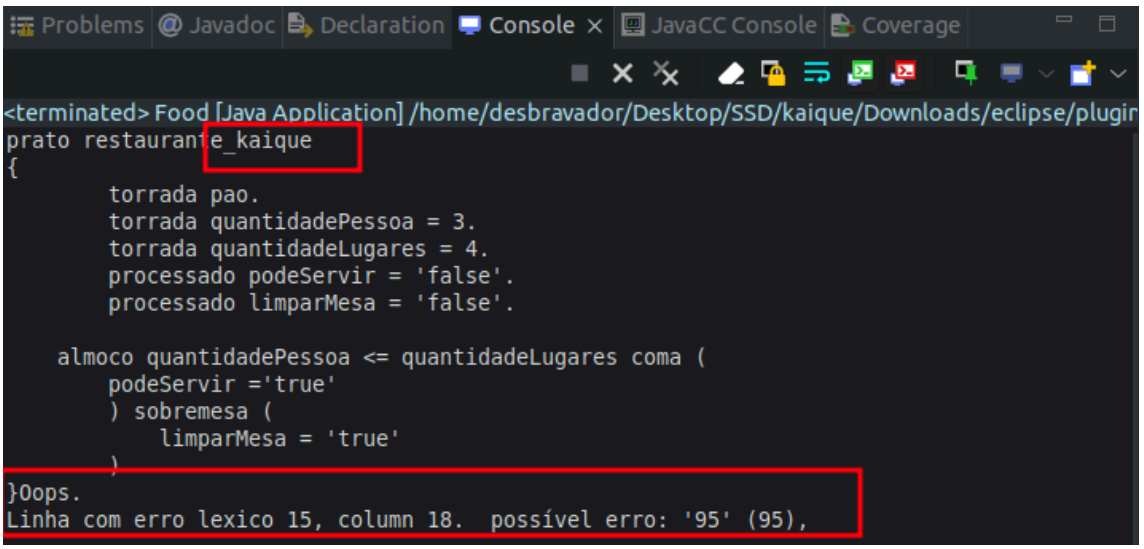
```
prato teste {
   torrada pao.
   torrada quantidadePessoa = 3.
   torrada quantidadeLugares = 4.
   processado podeServir = 'false'.
   processado limparMesa = 'false'.

   almoco quantidadePessoa <= quantidadeLugares coma (
```

```
        podeServir ='true'
    ) sobremesa (
        limparMesa = 'true'
    )
```

5.1 Executável aceito pela linguagem



5.2 Executável não aceita pela linguagem



6. Referências

[1] Alfred V. Aho, Ravi Sethi e Jeffrey D. Ullman. Compiladores: Princıpios, Técnicas e Ferramentas. LTC, 1995

[2]BARBOSA, Cynthia da S.; LENZ, Maikon L.; LACERDA, Paulo S. Pádua de; et al. **Compiladores**. Grupo A, 2021. *E-book.* ISBN 9786556902906. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556902906/. Acesso em: 5 abr. 2024.

[3]Vinicius , BRUM. “COMPILADOR PARA LINGUAGEM REVERSÍVEL JANUS.” 2017.Disponível em :https://app.uff.br/riuff/handle/1/5685 Acessado em: 5 de Abril de 2024.