P0DR40C0D3
Projeto Bimestral - Construção de Compiladores 2019/2

Angelo L. Bianchin[1], Fabiano F. Massignani[1], Jefferson A. Oliveira[1], Otavio Nascimento[1]

[1]Departamento Academico de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Medianiera – PR – Brazil

{bianchin, fabianofigueredo, oliveiraj, onascimento}@alunos.utfpr.edu.br

Abstract. *This paper aims to present the characteristics of three analyzes made by the compiler, lexic, syntactic and semantic. Similarly their development and use within a programming language that uses thematic terms related to snacks and food, in which all the sentences used were customized according to the term mentioned.*

Resumo. *Este trabalho tem como objetivo apresentar as características de três análises feitas pelo compilador, léxica, sintática e semântica. De mesma forma o desenvolvimento e emprego delas dentro de uma linguagem de programação que utiliza termos temáticos refentes a lanches e comidas, na qual todas as sentenças empregadas foram personalizadas em função do termo citado.*

1. Introdução

Linguagem[1] é o método que o homem utiliza para transmitir e comunicar suas ideias e sentimentos. Dentro dos estudos da linguística, são definidos vários tipos de linguagens e suas possíveis implementações. Linguagem de programação é uma linguagem escrita e formal que tem como objetivo especificar um determinado conjunto de instruções e regras utilizadas para construir programas (softwares). Essas regras são chamadas de linguagens fontes [Aho et al. 1995].

Dentro das linguagens de programação há dois níveis de classificação, alto e baixo nível. Esse nivelamento pode ser entendido como o nível de detalhamento do algoritmo para que ele execute uma determinada tarefa. Linguagens de alto nível utilizam uma abstração para facilitar o desenvolvimento do programa, já as de baixo nível se aproximam do alto detalhamento das linguagens de máquina.

Para que as linguagens computacionais sejam compreendidas e validadas é necessário a utilização de um compilador. Desta forma, um compilador é um programa que lê e traduz linguagens fontes para linguagem de máquina. Sendo assim o compilador atua como um tradutor, transcrevendo a linguagem original sem perder o contexto, porém, neste caso, também transforma o código de alto nível para baixo nível outrossim executa otimizações caso elas sejam necessárias. Criando assim programas equivalentes ao código de entrada.

Deste modo este trabalho tem como objetivo apresentar o desenvolvimento de uma linguagem de programação de acordo com as instruções propostas pela disciplina. Bem como demonstrar as etapas de análises executadas pelo compilador.

[1]"Linguagem", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2013, http://www.priberam.pt/dlpo/chave [consultado em 21-10-2013].

2. Análise léxica

A análise léxica possui duas etapas, em primeira instância o compilador executa uma varredura no código fonte removendo comentários e espaços em branco, sendo assim, executa uma otimização no código caso necessária. Na segunda etapa os analisadores realizam um processo de compilação onde executam a leitura do programa-fonte caractere por caractere, agrupam os caracteres em lexemas e produzem uma sequência de símbolos léxicos chamados de tokens [Maragon 2017]. Como ilustra a imagem abaixo:

Lexemas	Tokens
index	IDENTIFICADOR
=	OP_ATRIBUICAO
2	INT_LITERAL
*	OP_MULTIPLICAÇÃO
cont	IDENTIFICADOR
+	OP_SOMA
17	INT_LITERAL
;	PONTO_E_VIRGULA

Figura 1. Lexemas e tokens

2.1. Lexemas, Tokens e Padrões

São definidos três termos utilizados para a implementação de um analisador léxico: padrão, lexema e token. Padrões identificam a forma que os lexemas podem assumir em uma cadeia de caracteres, no caso de palavras reservadas, corresponde a sequência propriamente dita que constitui a palavra reservada. Já para os identificadores são os caracteres que formam propriamente os nomes das funções e variáveis. Já os lexemas são a sequência de caracteres reconhecidas pelos padrões. Por último, os tokens são palavras reservadas das linguagens, como mostra o exemplo a seguir:

Token	Padrão	Lexema	Descrição
`<const, >`	Sequência das palavras c, o, n, s, t	const	Palavra reservada
`<while, >`	Sequência das palavras w, h, i, l, e	while, While, WHILE	Palavra reservada
`<if, >`	Sequência das palavras i, f	If, IF, iF, If	Palavra reservada
`<=, >`	<, >, <=, >=, ==, !=	==, !=	
`<numero, 18>`	Dígitos numéricos	0.6, 18, 0.009	Constante numérica

Figura 2. Lexemas, tokens e padrões

3. Análise Sintática

O estudo da sintaxe busca validar a disposição das palavras dentro de frases e frases dentro de um texto. No caso das linguagens de programação a análise sintática tem como

objetivo identificar se o fluxo de tokens possui sentenças válidas em função da linguagem de programação [Maragon 2017].

Para realizar a análise léxica são construídos modelos baseados em gramáticas livres de contexto de forma a representar uma gramática formal que pode ser escrita através de algoritmos. Desda forma torna-se possível derivar todas as possíveis construções da linguagem. Assim como a análise léxica, a análise sintática possui termos específicos, classificados como:

- Símbolo: Elementos gráficos mínimos que compõem uma linguagem, exemplo: letras;
- Sentença: Conjunto ordenada de símbolos que formam uma string, exemplo: palavras;
- Alfabeto: Conjunto determinado de símbolos, exemplo: a, b, c, d, e...;
- Linguagem: Conjunto de sentenças, exemplo: flores, rosas;
- Gramática: Representação das regras para formação de uma linguagem.

Essa classificação pode ser compreendida em linguagem de programação segundo o exemplo de [Maragon 2017]:

- Alfabeto: w, h, i, l, e, +, 1, 2, 3
- Símbolos: 1, 5, +, w
- Sentença: while, 123, +1
- Linguagem: while, 123, +1

3.1. Backus-Naur

Aplicado quando as descrições de linguagens são necessárias, o Backus Normal Form (BNF) é um procedimento de notação metassintática usado para especificar a sintaxe das linguagens de programação de computador, conjuntos de comandos / instruções, formatação de documentos e protocolos de comunicação. Para o caso das linguagens livres de contexto, demonstra-se o exemplo utilizado por [Maragon 2017]:

```
Gramatica livre de contexto
G ({S, M, N}, {x,y}, P, S)
P {
    S  x | M
    M  MN | xy
    N  y
}
Forma de Backus-Naur
G = ({<S>, <M>, <N>}, {x,y}, P, <S>)
<S> ::= x | <M>
<M> ::= <M> <N> | xy
<N> ::= y
```

4. Análise Semântica

Responsável por verificar os aspectos referentes aos significados das instruções, a análise semântica complementa as duas análises anteriores. Sua execução se baseia em percorrer

a árvore sintática relacionando os identificadores com seus dependentes de acordo com a estrutura hierárquica definida. [Maragon 2017].

Durante a execução do processo, o compilador captura informações sobre o programa-fonte de forma que as fases subsequentes gerem o código objeto. Para isso, os tipos de dados são muito importantes, pois servem como notações dos valores para a linguagem de programação. Segundo [Santos and Langlois 2018]. a execução da análise semântica possui duas frentes:

- Estática: busca analisar todos os aspectos que possam ser identificador em tempo de compilação, dessa forma otimizando o tempo de execução.
- Dinâmica: verifica em tempo de execução em função de linguagens de programação que utilizam variáveis determinadas pelo contexto de uso, exemplo: PHP, LISP e etc.

5. P0DR40C0D3

Neste capítulo, será descrito a síntaxe do compilador Código P0dr40, suas funcionalidades e exemplos de códigos validos e inválidos.

5.1. Atribuições

Nesta seção, será descrito as sintaxes de atribuição como inicialização de bloco de comandos, fecho de instrução e atribuição de variável.

5.1.1. ”liga”

Sintaxe utilizada para iniciar um programa. Esta sintaxe é obrigatoria para inicialização.

5.1.2. ”prepara”

Sintaxe utilizada para iniciar um bloco de comandos. Esta sintaxe é obrigatorio para instrução de comandos e deve suceder a sintaxe ”liga”.

5.1.3. ”entrega”

Sintaxe utilizada para encerrar um bloco de comandos. Um exemplo da implementação das atribuições está descrita a seguir:

```
liga
    prepara
    <Comandos>
    entrega
```

5.1.4. ”prensou”

Sintaxe utilizada para fechar uma linha de comandos. A falta desta sintaxe pode acarretar erros.

```
liga
    prepara
    xburger cliente_0 prensou
entrega
```

5.1.5. ”cliente_”

Sintaxe utilizada para identificar uma variável. Todas as variáveis da linguagem devem preceder da sintaxe e podem conter 1 ou mais caracteres e números. Devem preceder do tipo primitivo, explicados na seção ”Tipos Primitivos”. Exemplo:

```
xburger cliente_0 prensou
```

5.1.6. ”adiciona”

Sintaxe utilizada para atribuir um valor ou caracter à uma variável. Exemplo:

```
xburger cliente_0 adiciona salada prensou
```

5.2. Tipos Primitivos

Nesta seção será descrita os tipos primitivos permitidos pelo P0DR40C0D3.

5.2.1. ”xburger”

Sintaxe para criar variáveis que permitem atribuir números inteiros (que aceite valores negativos e positivos). Exemplo:

```
liga
    prepara
    xburger cliente_0 prensou
    xburger cliente_1 adiciona 1 prensou
entrega
```

5.2.2. ”xtudo”

Sintaxe para criar variáveis que permitem atribuir 1 caracter ou mais.

```
liga
    prepara
    xburger cliente_0 prensou
    xburger cliente_1 adiciona salada prensou
entrega
```

5.2.3. ”xsalada”

Sintaxe para criar variáveis que permitem atribuir qualquer valor númerico inteiro ou decimal. Exemplo

```
    liga
        prepara
        xsalada cliente_0 prensou
        xsalada cliente_1 adiciona 10,0 prensou
    entrega
```

5.3. Condicional

Nesta seção será descrita as sintaxes para criação de condicionais para o compilador P0DR40C0D3.

5.3.1. ”tacomfome?”

Sintaxe para inicialização do bloco de condicional. Deve suceder com uma condicional e uma expressão, obrigatoriamente dentro de um bloco com parênteses. Exemplo:

```
liga
    prepara
    xtudo cliente_0 prensou
    xtudo cliente_1 prensou
    xburger cliente_2 prensou

    tacomfome? { {4 - {32 - 3}} + 5 == {4 - {32 - 3}} + 5 }(
        xsalada cliente_4 prensou
        cliente_2 adiciona 478513 prensou
        cliente_1 adiciona {4 - {32 - 3}} + 5 prensou
    )

    xburger cliente_03 prensou
    cliente_03 adiciona ABCDEFAGJ prensou
entrega
```

5.4. Laços de Repetição

Nesta seção será descrita as sintaxes para criação de laços de repetição para o compilador P0DR40C0D3

5.4.1. ”saboreando”

Sintaxe para inicialização do bloco de repetição **saboreando**. Deve ser inicializado com um identificador, seguido de condição de iteração. Exemplo:

```
liga
prepara
xtudo cliente_0 prensou
xtudo cliente_1 prensou
xburger cliente_2 adiciona 2 prensou
    saboreando
    {
    cliente_2 = 3 ate 8
    }
    (
    cliente_2 adiciona 1
        prensou
    )
entrega
```

5.4.2. ”fazer”

Sintaxe para inicialização do bloco de repetição **fazer**. Deve ser inicializado de instrução seguido de condição. Exemplo:

```
liga
prepara
xtudo cliente_0 prensou
xtudo cliente_1 prensou
xburger cliente_2 adiciona 2 prensou
    fazer
    (
        cliente_2 adiciona 1
        prensou
        )
    enquanto
        {
        cliente_2 <= 12
        }
entrega
```

6. Backus Naur

Grámatica convertida em Backus Naur:

```
< pricipal >        ::= < INICIOPROGRAMA > < INICIOBLOCO >  < bloco >* < FIMBLOCO >

< bloco >           ::= < decVariavel > | < expressao > |
```

```
                          < condicionalSe > | < repeticaoFor > | < repeticaoDo >

<decVariavel>        ::= < tipoDado > < atribuiVar > (< VIRGULA > < atribuiVAR >) *
                          <FIMINSTRUCAO>

<atribuiVar>         ::= < IDENTIFICADOR >
                          (( <ATRIBUICAO> (< CARACTERE > | < NUMERO >)) | <empty>)

< expressao >        ::= < IDENTIFICADOR > < ATRIBUICAO >
                          ((< CARACTERE >)(< CARACTERE >)* | < conta > ) < FIMINSTRUCAO >

< condicionalSe >    ::= < IF > < ABRECHAVE > < condicao > < FECHACHAVE >
                          < ABREPAR > < bloco >* < FECHAPAR >

< repeticaoFor >     ::= < FOR > < ABRECHAVE >(< IDENTIFICADOR >) < IGUAL > (< DIGITO > | < NUMERO >)
                          < PARA > (< DIGITO > | < NUMERO >) < FECHACHAVE > < ABREPAR >
                          < bloco >* < FECHAPAR >

< repeticaoDo >      ::= < DO > < ABREPAR > < bloco >* < FECHAPAR >
                          < WHILE > < ABRECHAVE > < condicao > < FECHACHAVE >

< tipoDado >         ::= < REAL > | < INTEIRO > | < LETRA >

< conta >            ::= < valores > ( < operacao > < valores >)*

< condicao >         ::= < conta > ( < comparacao > ) < conta >

< valores >          ::= < IDENTIFICADOR > | < DIGITO > | < NUMERO > | < contaComPresced >

< contaComPresced >  ::= < ABRECHAVE > < conta > < FECHACHAVE >

< operacao >         ::= < SOMA > | < SUBTRACAO > | < MULTIPLICACAO > | < DIVISAO >

< comparacao >       ::= < NEGACAO > |< IGUALIGUAL >
                          | < IGUAL > | < IGUALMAIOR > | < MAIOR >
                          | < IGUALMENOR > | < MENOR >
```

7. Exemplos de códigos

Dessa forma ilustra-se a seguir um exemplo de código válido dentro da linguagem descrita anteriormente:

```
liga prepara
    xtudo cliente_0 prensou
    xtudo cliente_1 prensou
    xburger cliente_2 prensou

    tacomfome? { {4 - {32 - 3}} + 5 == {4 - {32 - 3}} + 5 }(
        xsalada cliente_4 prensou
        cliente_2 adiciona 478513 prensou
        cliente_1 adiciona {4 - {32 - 3}} + 5 prensou
    )

    xburger cliente_03 prensou
    cliente_03 adiciona ABCDEFAGJ prensou
entrega
```

```
liga
    prepara
    xtudo cliente_0 prensou
    xtudo cliente_1 adiciona 3.0 prensou
    xburger cliente_2 adiciona 2 prensou
        saboreando cliente_2 = 3 ate 8
            cliente_2 adiciona 1 prensou
entrega
```

8. Conclusão

Embora o compilador faça todo o trabalho duro de converter o nível das linguagens, a criação de uma linguagem de programação requer o entendimento das suas etapas de funcionamento. No presente trabalho foram apresentadas as etapas léxicas, sintáticas e semânticas. De mesma forma a apresentação da linguagem P0DR40C0DE que utiliza termos cômicos com a temática de lanches e comidas para a construção dos componentes da linguagem.

Referências

Aho, A. V., Sethi, R., and Ulman, J. D. (1995). *Compiladores princípios, técnicas e ferramentas*. Rio de Janeiro: Livros Técnicos e Científicos, 1th edition.

Maragon, J. D. (2017). *Compiladores para Humanos*. Gitbook, 1th edition.

Santos, P. R. and Langlois, T. (2018). *Compiladores da teoria à prática*. Grupo Editorial Nacional, 1th edition.