# Analisador Sintático

**Trabalho de Tradutores - parte 3**

João Gabriel Lima Neves - 15/0131992

Profª. Cláudia Nalon

Setembro de 2020

## 1 Introdução

Entender o processo de tradução de um programa é essencial para a formação de um cientista da computação. O curso de Tradutores ministrado na Universidade de Brasília serve justamente a providenciar aos estudantes do curso de Ciência da Computação o conjunto de habilidades para compreender o processo de compilação de um programa. Desta forma, o trabalho do curso consistirá em 6 etapas aonde será desenvolvido de um tradutor sendo estas: Escolha do Tema, Analisador Léxico, Analisador Sintático, Analisador Semântico, Gerador de Código Intermediário e Apresentação do Trabalho. Este relatório tratara da primeira etapa.

## 2 Motivação

Atualmente, a física computacional possui atuação muito mais abrangente que a tradicional. O conhecimento em Ciência e Engenharia da Computação é utilizado como ferramenta para os avanços tanto em física teórica como experimental, ao mesmo tempo em que conceitos da Física são aplicados à Teoria da computação [2]. Ao mesmo tempo, o mercado de jogos eletrônicos teve um crescimento bastante significativo nas ultimas décadas sendo que em 1995 contávamos com 100 MM de jogadores (Gamers, do inglês) e passamos para 2.6 bilhões de jogadores ativos em torno do globo [1]. Uma necessidade comum nessas duas áreas é a de representar forças, velocidades e posições de objetos em um plano. Dessa forma é interessante para essas duas áreas a existência de uma estrutura de dados dentro de uma linguagem que permita representar vetores.

Linguagens como o `C#`[1] providenciam ao programador a capacidade de poder abstrair varias operações comuns relacionadas a manipulação de vetores como soma, subtração, normalização e distancia entre dois pontos por exemplo, de tal forma ela é a linguagem

[1] https://docs.microsoft.com/pt-br/dotnet/csharp/

usado pelo Unity, uma Game Engine bastante utilizada no mercado para o desenvolvimento de jogos eletrônicos. A linguagem `C` por outro lado, não oferece esse tipo de abstração.

## 3 Objetivo do Projeto

Este trabalho visa garantir operações básicas entre pares ordenados para facilitar e optimizar operações comuns relacionadas a manipulação de vetores. Dessa forma ele se propõem a implementar um tradutor para uma versão simplificada da linguagem `C` que suporte comandos de leitura e escrita e chamadas de sub-rotinas, operações de controle condicional de fluxo e laços de repetição, operações com inteiros e números de ponto flutuante e, como uma adição ao `C`, pares ordenados(Vectors) com operações de adição e subtração entre vetores físicos, multiplicação de entre vetores, multiplicação entre inteiros ou reais e vetores, calcular a distancia entre dois vetores e normalização de um vetor. Um exemplo do que seria um código dessa linguagem pode ser visto abaixo:

```
1  Vector2 vect1;
2  Vector2 vect2;
3  Vector3 vect3;
4
5  vect1 = <1.0 , 2.7>;
6  vect2 = <3.5 , 4.0>;
7  vect3 = vect1 + vect2;
8
9  write(vect3[0])
10 write(vect3[1])
11 write(vect3)
```

```
>> 4.5
>> 6.7
>> <4.5 , 6.7>
```

## 4 Gramatica

A forma utilizada no curso, e que será utilizada neste trabalho, de especificar uma sintaxe para uma linguagem é a de construir uma gramática livre de contexto. A gramática da linguagem que será desenvolvida neste projeto pode ser encontrada abaixo:

1. $prog \rightarrow declarationList$
2. $declarationList \rightarrow declarationList\ declaration \mid declaration$
3. $declaration \rightarrow variableDeclaration \mid functionDeclaration$
4. $variableDeclaration \rightarrow$ TYPE **ID ;**
5. $functionDeclaration \rightarrow$ TYPE **ID (** $params$ **)** $compoundStmt$

6. $params \rightarrow paramList \mid$ **void**
7. $paramList \rightarrow paramList$ **,** $param \mid param$
8. $param \rightarrow$ TYPE **ID**
9. $compoundStmt \rightarrow$ **{** $localDeclarations\ stmtList$ **}**
10. $localDeclarations \rightarrow localDeclarations\ variableDeclaration \mid \varepsilon$
11. $stmtList \rightarrow stmtList\ stmt \mid \varepsilon$
12. $stmt \rightarrow expressionStmt \mid conditionalStmt \mid iterationStmt \mid returnStmt \mid IOStmt \mid vectorStmt$
13. $stmt \rightarrow localDeclarations \mid assingStmt$
14. $expressionStmt \rightarrow simpleExpression$ **;**
15. $conditionalStmt \rightarrow$ **if (** $simpleExpression$ **)** $compoundStmt \mid$
    **if (** $simpleExpression$ **)** $compoundStmt$ **else** $compoundStmt$
16. $iterationStmt \rightarrow$ **while (** $simpleExpression$ **)** $compoundStmt$
17. $returnStmt \rightarrow$ **return** $expression$ **; |** **return ;**
18. $IOStmt \rightarrow\ read(var)$**;** $\mid write(var)$ **;**
19. $vectorStmt \rightarrow\ normalize(Vector2)$**;** $\mid distance(Vector2, Vector2)$ **;**
20. $assingStmt \rightarrow\ var = expression\ ;$
21. $expression \rightarrow opExpression$
22. $var \rightarrow$ **ID**
23. $simpleExpression \rightarrow opExpression\ relop\ opExpression \mid opExpression$
24. $relop \rightarrow$ **COMPARABLES**
25. $opExpression \rightarrow opExpression\ operators\ term \mid term$
26. $operators \rightarrow$ **+** | **-** | ***** | **/** | || | &&
27. $term \rightarrow term\ factor \mid factor$
28. $factor \rightarrow$ **(** $expression$ **)** $\mid var \mid call \mid$ **INT** | **FLOAT** | $Vector2$ | **BOOL**
29. $call \rightarrow$ **ID (** $args$ **)**
30. $Vector2 \rightarrow <$ **INT , INT** $> \mid <$ **FLOAT , FLOAT** $>$
31. $args \rightarrow argList \mid \varepsilon$

32. $argList \rightarrow argList$ **,** $expression \mid expression$

**ID** = $letter\ (letter|digit)^*$

**STRING** = $"(letter|digit) * "$

**BOOL** = $true|false$

**INT** = $digit\ digit^*$

**FLOAT** = $digit\ digit^*.\ digit^*$

**TYPE** = $int|float|char|vector|bool|void$

**COMPARABLES** = $<= \mid == \mid >= \mid > \mid < \mid != =$

$letter = a \mid ... \mid z \mid A \mid ... \mid Z$

$digit = 0 \mid ... \mid 9\ \backslash t$

Símbolos especiais: + - * / < <= > >= == != = , ; ( ) [ ] { } # “ ”

## 4.1 Revisões da gramatica

Do documento do trabalho anterior a este foram feitas tais revisões a gramatica:

1. Regra 14 transiciona para "simpleExpression"ao inves de "expression"para resolver problemas de redução.
2. "type"deixou de ser uma regra da gramatica e passou a ser o token "TYPE", pois na implementação do analisador sintático percebeu-se que faz mais sentido o tipo de variável ser implementado dessa maneira.
3. Regra 20 "assingStmt"adicionada a linguagem, serve para fazer a operação de atribuição.
4. Transição para "assingStmt"adicionada a regra 13.
5. Transição para "localDeclarations"adicionada a regra 13 para permitir que declarações locais de variaveis possam ser feitas em qualquer da função e não só no inicio da função.
6. Regra 21 alterada para conter apenas a transição para "opExpression"já que a função de atribuição foi realocada para a regra 20.

## 5 Semantica

1. A linguagem apresenta escopo estático.
2. As unicas converções implícitas serão de int para float, com a casa decimal do int sendo considerada como 0, e de float de para int, que sempre descartara a parte decimal.
3. A passagem de parametros e as atribuições se darão sempre por copia.
4. O ponto inicial de execução será sempre o método main, não sendo executado sem ele.
5. A primitiva vector só pode ser composta por dois ints ou dois floats.
6. Todos os outros construtos serão avaliados como em C

## 6 Analisador Léxico

O analisador léxico a ser utilizado foi desenvolvido com o uso da ferramenta **flex**, que, uma vez definida as expressões regulares para os símbolos da nossa linguagem, gera o arquivo c++ que contém o código que fara a análise léxica desses símbolos definidos.

Ao ser executado em um arquivo txt contendo o código na linguagem desenvolvida, o analisador retorna todos os os símbolos (sejam eles identificadores, comentários, números inteiros, floats, vetores, strings, tipos, operadores, etc) na ordem que ele encontra ao fazer a analise. Ele também ao encontrar algum simbolo que não se encaixa em nenhuma da expressões regulares que identificam símbolos da linguagem retorna o simbolo não identificado na linha em que o encontrou, dessa forma ajuda desenvolvedores a facilmente identificar a presença do simbolo errado e remove-lo.

## 7 Analisador Sintático

O analisador sintático da linguagem foi desenvolvido utilizando-se a ferramenta **bison**. Para utiliza-la teve-se que escrever um código "analisadorSintatico.y"aonde as regras da linguagem foram listadas e utilizadas para montar duas estruturas que searão utilizadas na pelo analisador semântico a ser implementado.

A primeira estrutura ,a tabela de símbolos, foi implementada como um hash com a biblioteca "uthash.h"recomendada pela professora. Essa tabela armazena todas as declarações de funções e variáveis encontradas no código. Na implementação cada elemento do hash contém informações como o nome do simbolo, se ele se trata de uma função ou variável, o tipo (de retorno no caso do simbolo ser uma função) da variável e o escopo em que se encontra o simbolo, ele também contém um token único por simbolo que é formado a partir da concatenação das informações listadas anteriormente.

A segunda estrutura é a arvore sintática, que representa o fluxo de execução das expressões da linguagem e que será utilizada pelo analisador semântico para executa-lo. Ela é formada por nós possuem ponteiros para outros dois nós, um da direita e o outro da esquerda, e possui informações como o nome do simbolo, o tipo de variavel do simbolo e o tipo do nó("FUNCTION", "VARIABLE", "OPERATOR", "VECTOR", "IO", "CONDITIONAL", "PARAMS"ou "STATEMENT").

Para a politica de tratamento de erros sintáticos esta sendo impressa, a cada erro sintático que o **bison** detecta durante a execução, uma mensagem de erro com a linha aonde o erro foi encontrado seguido da mensagem de erro do **bison**. Foi escolhido esse tratamento já que as mensagens de erro do **bison** já são bem explicativas do problema sintático, sendo o seu único problema não apresentar a linha do erro, essa parte da mostrar a linha foi implementada pois ajuda o desenvolvedor a identificar mais rápido aonde se encontra o erro.

## Referências


[1] João Victor Oliveira Eduardo Henrique Viva, Matheus de Souza Amorim. TendÊncias no mercado de games e sua importÂncia. `http://revista.faqi.edu.br/index.php/seminario/article/view/400` [Online; accessed 19-Março-2019].

[2] wiki. Física computacional. `https://pt.wikipedia.org/wiki/F%C3%ADsica_computacional` [Online; accessed 19-Março-2019].