# Desenvolvimento de Um Compilador - Análise Sintática

**Pedro H. R. Souza**
Departamento de Ciência da Computação – Universidade Federal de Lavras(UFLA)
37.200-000 – Lavras – MG – Brasil
phramos@computacao

# 1. Introdução

Um compilador é um programa capaz de processar uma dada entrada em texto e a transformar em algo que um computador possa compreender. O processo de compilação possui diversas etapas, entre elas: análise léxica; análise sintática; análise semântica; geração de código intermediário; otimização de código; e geração de código. A análise sintática é uma fase responsável por ler caractere por caractere da entrada e os converter em diversos tokens agrupados de maneira lógica e coerente. Este trabalho tem por objetivo implementar um analisador sintático simplificado de uma linguagem de programação.

# 2. Metodologia

## 2.1. Ferramentas e Tecnologias Utilizadas

Este trabalho utilizou diversas tecnologias para o desenvolvimento de um analisador sintático de uma linguagem de programação. Dentre as tecnologias utilizadas, as principais e mais significativas foram: *antlr4*; *Java*; *Intellij IDEA*. Informações mais detalhadas sobre essas ferramentas podem ser obtidas nas subseções deste tópico.

### 2.1.1. ANTLR4

O antlr 4 é um poderoso gerador de conversores para leitura, processamento, execução, e tradução de arquivos em texto ou binários. Atualmente, o antlr é amplamente utilizado se construir linguagens e ferramentas de programação.

Este trabalho utilizou o *antlr4* para se converter definições de uma gramática em notação padrão dentro de um arquivo .g4 para classes Java que puderam ser utilizadas na implementação de um analisador sintático para a linguagem de programação.

### 2.1.2. Linguagem Java

Java é uma linguagem de programação orientada à objetos que pode ser utilizada para os mais diversos propósitos, entre eles até mesmo o desenvolvimento de um compilador.

Este trabalho utilizou classes geradas em Java por meio do antlr4 para implementar um analisador sintático de uma linguagem de programação.

### 2.1.3. Intellij Idea

Intellij IDEA é uma robusto ambiente integrado de desenvolvimento para a linguagem de programação JAVA. Ele é capaz de realizar análises de código estaticamente para poder aumentar a produtividade do desenvolvimento de programas escritos em JAVA.

Este trabalho utilizou o IntelliJ Idea para escrever a gramática do analisador léxico e o analisador sintático em um arquivo .g4.

## 2.2. Estrutura do Projeto

Este trabalho foi desenvolvido dentro em uma estrutura proposta pelo autor deste relatório. Um esquema resumido da estrutura deste  projeto encontra-se abaixo:

--Pasta raiz do projeto contendo todas as subpastas, arquivos de código, e
--bibliotecas de terceiros.
*/*

--Arquivos em texto utilizados na execução do programa.
**arquivos**
--Arquivos de testes utilizados para validação da implementação.
**gen**
--Arquivos gerados pelo antlr4 utilizados na implementação da solucão.
**gramatica**
--Arquivos .g4 que são utilizados como base da geração de código pelo antlr4.
**lib**
--Bibliotecas de terceiros utilizados no projeto, tais como o antlr4.
**src**
--Arquivos.java

## 2.3. Soluções Propostas

A solução proposta foi baseada no arquivo base de um analisador sintátito disponibilizado pelo professor Rafael Serapilha Durelli do Departamento de Ciência da Computação da Universidade Federal de Lavras. Mais detalhes da implementação pode ser verificada no arquivo CMinus.g4 disponibilizado juntamente ao código fonte da solução

# 3. Testes

Foi desenvolvido dois arquivos para auxiliar nos testes e validação da gramática. mais detalhes a respeito dos testes podem ser encontrados nas subseções desse tópico.

## 3.1. Testes de Sucesso

Foram realizados diversos testes de sucesso para validar as principais regras sintáticas implementadas

O arquivo com o código completo pode ser verificado a seguir:

```
struct fauno{
  int altura;
  float peso;
  char sexo = 'M';
};

int variavelglobal;

int main(char args) {
  float a = 666;
  float b;
  float c = 10.0E56;


  while (a != b) {
      if (a == b) {
          a = b - 1;
      } else {
          b = a + 1;
      }
  }

  c = a + b - 35;

  struct fauno2{
      int altura;
      float peso;
      char sexo = 'M';
  };

  return 0;
}
```

## 3.2. Testes de Erros

Foram definidos 10 arquivos contendo erros sintáticos para validar a implementação da sintaxe da linguagem C-.

### 3.2.1. teste_errado_colchete_main.c

Trecho de código:

```
int main(char args) {
  return 0;
/*ERRO AQUI - falta um fecha chaves*/
```

Resultado:

```
line 5:0 missing '}' at '<EOF>'

Process finished with exit code 0
```

### 3.2.2. teste_errado_colchete_main.c

Trecho de código:

```
int main(char args) {
  /*ERRO - Falta parentese aqui*/
  if args) {
      return 0;
  }
}
```

Resultado:

```
line 3:7 missing '(' at 'a'

Process finished with exit code 0
```

### 3.2.3. teste_errado_ponto_e_virgula.c

Trecho de código:

```
int main(char args) {
  /*ERRO AQUI - falta o ;*/
  return 0
}
```

Resultado:

```
line 4:0 mismatched input '}' expecting {'>', ';', '=', '<', '==', '<=',
'>=', '!=', '+', '-', '*', '/'}

Process finished with exit code 0
```

### 3.2.4. teste_errado_tipo_se_identificador1.c

Trecho de código:

```
int main(char ) {
  return 0;
}
```

Resultado:

```
line 1:14 missing Identifier at ')'

Process finished with exit code 0
```

### 3.2.5. teste_errado_tipo_se_identificador2.c

Trecho de código:

```
int main(char args ) {
   int 0;
   return 0;
}
```

Resultado:

```
line 2:8 mismatched input '0' expecting Identifier

Process finished with exit code 0
```

### 3.2.6. teste_errado_identificador_identificador.c

Trecho de código:

```
int main(char args ) {
   int int;
   return 0;
}
```

Resultado:

```
line 2:8 mismatched input 'int' expecting Identifier

Process finished with exit code 0
```

### 3.2.7. teste_errado_argumentos1.c

Trecho de código:

```
int main(char args,) {
   return 0;
}
```

Resultado:

```
line 1:19 mismatched input ')' expecting {'int', 'float', 'char'}

Process finished with exit code 0
```

### 3.2.8. teste_errado_argumentos2.c

Trecho de código:

```
int qualquercoisa(char args, beluga) {
    return 0;
}
```

Resultado:

```
line 1:29 missing {'int', 'float', 'char'} at 'beluga'

Process finished with exit code 0
```

### 3.2.9. teste_errado_struct1.c

Trecho de código:

```
struct fauno {
}
```

Resultado:

```
line 3:0 mismatched input '}' expecting {'int', 'float', 'char'}

Process finished with exit code 0
```

### 3.2.10. teste_errado_struct2.c

Trecho de código:

```
struct pessoa {
    int idade;
    list associados;
};
```

Resultado:

```
line 3:4 extraneous input 'list' expecting {'int', 'float', 'char', '}'}

Process finished with exit code 0
```

# 4. Conclusão

Através da implementação do analisador sintático da linguagem proposta por este trabalho pôde-se concluir que é possível realizar a implementação de um analisador sintático para a linguagem C- utilizando o antlr4 juntamente aos conhecimentos obtidos na disciplina de compiladores associado ao livro texto da disciplina.

# 5. Referencial Teórico

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman(2006) “Compilers: Principles, Techniques, and Tools”.
http://www.antlr.org/, Fevereiro de 2016.
https://www.oracle.com/java/index.html, Fevereiro de 2016.
http://compilerdesigndetails.blogspot.com.br/2012/02/v-behaviorurldefaultvmlo.html, Fevereiro de 2016.
http://www.antlr.org/api/Java/org/antlr/v4/runtime/Parser.htm,l Fevereiro de 2016.