

Explicação Etapa 1

Analizador Léxico (Lexer)

O analisador léxico é a primeira etapa da compilação. Sua função é ler o código-fonte como uma sequência de caracteres e transformá-lo em tokens, que são unidades léxicas significativas como palavras-chave (int, float), identificadores (idade, nome), operadores (=, +) e símbolos (;, ,).

Além disso, o léxico também:

- Ignora espaços, tabulações e comentários.
- Detecta erros léxicos, como caracteres inválidos ou identificadores malformados (ex: 2variavel).
- Classifica cada token com seu tipo e posição (linha e coluna), o que ajuda na análise sintática.

Exemplo de entrada:

```
int idade = 30;
```

Tokens gerados:

- INT, IDENTIFICADOR(idade), IGUAL, NUM_INT(30), PVIRG

Analizador Sintático (Parser)

O analisador sintático é a **segunda etapa da compilação**. Ele recebe os tokens do léxico e **verifica se a sequência está de acordo com a gramática da linguagem** — ou seja, se a estrutura do código está correta.

Ele trabalha como um verificador de **ordem e combinação**: não basta que os tokens estejam certos, eles precisam **estar na ordem certa**.

Funções do parser:

- Identifica **declarações válidas** (como int idade = 30;).
- Detecta **erros de sintaxe**, como int = 30;, que é inválido porque não tem um identificador antes do =.
- Garante que regras como **fim com ponto e vírgula, lista separada por vírgulas e tipos permitidos** estejam sendo respeitadas.

Estrutura dos Arquivos

- **AnalizadorLexico.flex**
Arquivo que define o analisador léxico. Nele estão especificados os padrões das expressões regulares que reconhecem palavras-chave (int, float, char), identificadores, números, literais e operadores. Também trata erros léxicos, como identificadores inválidos que começam com números.
- **AnalizadorSintatico.cup**
Contém a gramática usada pelo analisador sintático gerado com CUP. Define os tipos, estruturas de declaração, lista de identificadores, atribuições e valores. É responsável por validar a estrutura das declarações conforme as regras da linguagem.
- **AnalizadorLexico.java e AnalizadorSintatico.java**
Arquivos gerados automaticamente pelas ferramentas JFlex e CUP, a partir dos arquivos .flex e .cup. Esses arquivos implementam os analisadores propriamente ditos.
- **sym.java**
Classe que contém os símbolos terminais utilizados pela gramática. Também é gerada automaticamente pelo CUP.
- **TesteLexico.java**
Classe de teste utilizada para verificar apenas o analisador léxico. Lê o arquivo teste.txt e imprime os tokens identificados com seus valores, linha e coluna.
- **teste.txt**
Arquivo de entrada com exemplos de declarações de variáveis. É usado para testar o funcionamento dos analisadores.
- **Arquivos .class**
Arquivos compilados das classes Java (.java) para execução no interpretador da JVM.
- **jflex-full-1.8.1.jar e java-cup-11b.jar**
Bibliotecas necessárias para a geração e execução dos analisadores. São usadas para processar os arquivos .flex e .cup.

Como Compilar e Executar o Projeto

Pré-requisitos

- Java JDK instalado (recomendado: versão 8 ou superior)
- Terminal ou prompt de comando (cmd, PowerShell, etc.)
- Arquivos java-cup-11b.jar e jflex-full-1.8.1.jar disponíveis na mesma pasta do projeto

Etapa 1 – Gerar o Analisador Léxico com JFlex

```
java -jar jflex-full-1.8.1.jar AnalisadorLexico.flex
```

Etapa 2 – Gerar o Analisador Sintático com CUP

```
java -jar java-cup-11b.jar -parser AnalisadorSintatico -symbols sym  
AnalisadorSintatico.cup
```

Etapa 3 – Compilar os Arquivos Java

```
javac -cp ".;java-cup-11b.jar" AnalisadorLexico.java AnalisadorSintatico.java sym.java  
TesteLexico.java
```

Etapa 4 – Rodar o Analisador Léxico

```
java -cp ".;java-cup-11b.jar" TesteLexico
```

Etapa 5 – Rodar o Analisador Sintático Completo

```
java -cp ".;java-cup-11b.jar" AnalisadorSintatico
```

Exemplos de Entradas Válidas

```
int x;
```

```
float y = 2.5;  
char c = 'A';  
char nome = "Ana";  
int a, b, c;
```

Exemplos de Entradas Inválidas

```
double d;      // tipo não suportado  
int = 30;      // faltando identificador  
123abc = 10;   // identificador inválido
```

Explicação Etapa 2

Esta etapa do trabalho tem como objetivo desenvolver os analisadores léxico e sintático para o reconhecimento dos comandos de seleção `if` e `switch-case`, no estilo da linguagem C. A análise será realizada com o uso das ferramentas JFlex (analisador léxico) e JavaCUP (analisador sintático).

Requisitos de Sintaxe

Os comandos que devem ser reconhecidos nesta etapa são:

- Comando if:
 - - Sintaxe: `if(condição) { <comandos> } else { <comandos> }`
 - - A condição utiliza operadores relacionais (`==`, `!=`, `<`, `>`, `<=`, `>=`).
 - - Os comandos são expressões matemáticas simples terminadas por ponto e vírgula.
 - - O bloco else é opcional.
- Comando switch-case:
 - - Sintaxe: `switch(variável) { case id: { <comandos> break; } default: { <comandos> } }`
 - - Cases podem ser múltiplos ou únicos.
 - - O bloco default é opcional.

Analizador Léxico (.flex)

O analisador léxico foi construído com base na especificação das expressões regulares e estruturas da linguagem C. Ele reconhece identificadores, palavras-chave, operadores relacionais e aritméticos, símbolos, literais e ignora espaços em branco e comentários.

Analizador Sintático (.cup)

O analisador sintático define a gramática para reconhecer os comandos `if` e `switch-case`, seguindo a sintaxe e semântica da linguagem C. Foram utilizados não-terminais como `comando_if`, `comando_switch`, `condicao`, `comando`, entre outros. Também foi implementado suporte à precedência dos operadores matemáticos.

Testes Realizados

Foram realizados testes com arquivos de entrada válidos e inválidos para garantir a robustez dos analisadores.

Exemplo de código válido:

```
1 int x = 5, y;  
2 if (x > 2) {  
3     y = x + 1;  
4 } else {  
5     y = x - 1;  
6 }  
7
```

PROBLEMAS 0/0 SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

I:\Meu Drive\ULBRA\ULBRA SEMESTRES\Ulbra 2025-01\Compiladores\Trabalho Final - Criação de um Analisador Léxico e Sintático - Ferramentas de Análise\Etapa 2>java -cp ".;java-cup-11b.jar;jflex-full-1.8.1.jar" AnalisadorSintatico
? Código válido.

Exemplo de código inválido:

```
1 int x = 55a, y;  
2 if (x > 2) {  
3     y = x + 1;  
4 } else {  
5     y = x - 1;  
6 }  
7
```

PROBLEMAS 1/1 SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

Léxico e Sintático - Ferramentas de Análise\Etapa 2>java -cp ".;java-cup-11b.jar;jflex-full-1.8.1.jar" AnalisadorSintatico
Erro léxico: identificador inválido: 55a
Syntax error
instead expected token classes are [CHAR_LITERAL, STRING_LITERAL]
Couldn't recover from previous error(s)
Exception in thread "main" java.lang.Exception: Can't recover from previous error(s)
at java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:392)
at java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:539)
at java_cup.runtime.lr_parser.parse(lr_parser.java:731)
at AnalisadorSintatico.main(AnalisadorSintatico.java:211)

Como Compilar e Executar o Projeto

Pré-requisitos

- Java JDK instalado (recomendado: versão 8 ou superior)
- Terminal ou prompt de comando (cmd, PowerShell, etc.)
- Arquivos java-cup-11b.jar e jflex-full-1.8.1.jar disponíveis na mesma pasta do projeto

Etapa 1 – Gerar o Analisador Léxico com JFlex

java -jar jflex-full-1.8.1.jar AnalisadorLexico.flex

OBS: após essa etapa, é necessário inserir no AnalisadorLexico.java o import java_cup.runtime.Symbol; na primeira linha, conforme imagem abaixo.

```
1 // DO NOT EDIT
2 // Generated by JFlex 1.8.1 http://jflex.de/
3 // source: AnalisadorLexico.flex
4 import java_cup.runtime.Symbol;
5
```

Etapa 2 – Gerar o Analisador Sintático com CUP

java -jar java-cup-11b.jar -parser AnalisadorSintatico -symbols sym
AnalisadorSintatico.cup

Isso vai gerar os seguintes arquivos:

- AnalisadorSintatico.java
- sym.java

Etapla 3 – Criar o arquivo de teste

Crie um arquivo teste.txt com código a ser testado, por exemplo:

```
int x = 5, y;  
if (x > 2) {  
    y = x + 1;  
} else {  
    y = x - 1;}  

```

Etapla 4 – Compilar os Arquivos Java

```
javac -cp ".;java-cup-11b-runtime.jar" *.java
```

Etapla 5 – Rodar o Analisador Léxico

```
java -cp ".;java-cup-11b-runtime.jar" AnalisadorSintatico
```

Com isso o retorno deve ser esse

? Código válido.

Explicação Etapa 3

A Etapa 3 do trabalho tem como objetivo completar a construção do compilador incluindo todos os tipos de comandos da linguagem C definidos no projeto, especialmente os comandos de repetição while e for, além do refinamento de switch-case com múltiplos case seguidos. Esta etapa visa a validação completa da estrutura sintática, integrando as funcionalidades das etapas 1 e 2 com novos elementos gramaticais.

O analisador léxico foi responsável por reconhecer os **tokens básicos** usados nos comandos de repetição. Foram implementadas expressões regulares para os seguintes elementos:

- **Palavras-chave:** while, for
- **Parênteses e chaves:** () { }
- **Operadores de comparação:** ==, !=, <, >, <=, >=
- **Operadores aritméticos:** +, -, *, /
- **Operador de atribuição:** =

- **Delimitadores:** ; (ponto e vírgula)
- **Identificadores:** nomes de variáveis válidos
- **Números inteiros e reais**

Melhorias Implementadas na Etapa 3

Durante o desenvolvimento da Etapa 3, foi implementada uma melhoria importante no analisador sintático: a capacidade de reconhecer múltiplos case consecutivos que apontam para o mesmo bloco de comandos, conforme a sintaxe real da linguagem C.

Anteriormente, a gramática apenas reconhecia case isolados, o que limitava a construção de estruturas switch mais complexas. Após a melhoria, o analisador passou a aceitar múltiplos rótulos case para um mesmo conjunto de comandos, ou seja, vários pontos de entrada que compartilham o mesmo bloco.

Requisitos de Sintaxe

Comando while

```
while (condição) {  
  
    <comandos matemáticos>;  
  
}
```

Gramática

```
comando_while ::= WHILE ABRE_PAREN condicao FECHA_PAREN ABRE_CHAVE  
comandos FECHA_CHAVE
```

```
{: System.out.println("Reconhecido: WHILE"); :};
```

Comando for

```
for (int i = 0; i < 10; i = i + 1) {  
  
    <comandos matemáticos>;  
  
}
```

Gramática

```
comando_for ::= FOR ABRE_PAREN atribuicao PVIRG condicao PVIRG incremento  
FECHA_PAREN ABRE_CHAVE comandos FECHA_CHAVE
```

```
{: System.out.println("Reconhecido: FOR"); :};
```


Testes Realizados

Foram realizados testes com arquivos de entrada válidos e inválidos para garantir a robustez dos analisadores.

Exemplo de código válido:

```
1  int i;  
2  int contador = 0;  
3  
4  for (i = 0; i < 5; i = i + 1) {  
5      while (contador < 3) {  
6          contador = contador + 1;  
7      }  
8  }  
9
```

PROBLEMAS 0/0 SAÍDA CONSOLE DE DEPURACÃO **TERMINAL** PORTAS

Focar na pasta no explorador (ctrl + clique) (AnalizadorSintatico.java:300)

```
I:\Meu Drive\ULBRA\ULBRA\ULBRA SEMESTRES\Ulbra 2025-01\Compiladores\Trabalho Final - Criação de um Analisador Léxico e Sintático - Ferramentas de Análise\Etapa 3 - Final - Copia>java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalizadorSintatico  
Iniciando análise sintática...  
Reconhecido: WHILE  
Reconhecido: FOR  
Código válido.
```

```
1  int i;  
2  for (i = 0; i < 10; i = i + 1) {  
3      i = i * 2;  
4  }  
5
```

casos_caso Aa Nenhum resultado

PROBLEMAS 0/0 SAÍDA CONSOLE DE DEPURACÃO **TERMINAL** PORTAS

```
I:\Meu Drive\ULBRA\ULBRA\ULBRA SEMESTRES\Ulbra 2025-01\Compiladores\Trabalho Final - Criação de um Analisador Léxico e Sintático - Ferramentas de Análise\Etapa 3 - Final - Copia>java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalizadorSintatico  
Iniciando análise sintática...  
Reconhecido: FOR  
Código válido.
```

```
1 char letra;  
2 int opcao = 2;  
3  
4 switch (opcao) {  
5     case 1: {  
6         letra = 'A';  
7         break;  
8     }  
9     case 2: {  
10        letra = 'B';  
11        break;  
12    }  
13    default: {  
14        letra = 'Z';  
15    }  
16 }  
17
```

PROBLEMAS 2/2 SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL PORTAS

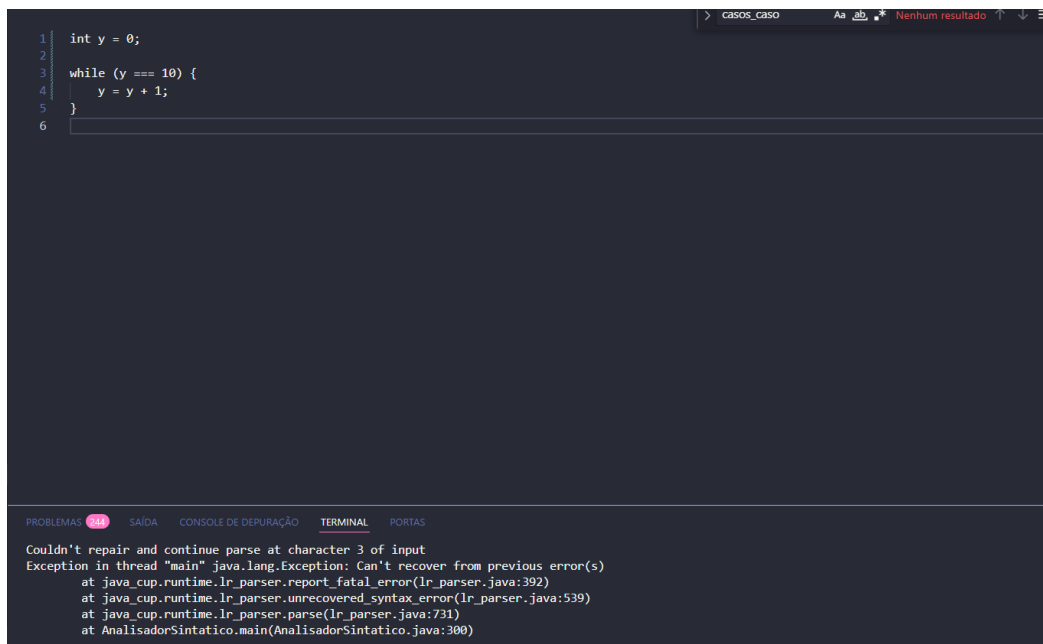
I:\Meu Drive\ULBRA\ULBRA\ULBRA SEMESTRES\Ulbra 2025-01\Compiladores\Trabalho Final - Criação de um Analisador Léxico e Sintático - Ferramentas de Análise\Etapa 3 - Final - Cópia>java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalisadorSintatico
Iniciando análise sintática...
Reconhecido: CASE
Reconhecido: CASE
Reconhecido: DEFAULT
Reconhecido: SWITCH
Código válido.

```
1 int x = 0;  
2  
3 while (x < 10) {  
4     x = x + 1;  
5 }  
6
```

PROBLEMAS 2/2 SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL PORTAS

I:\Meu Drive\ULBRA\ULBRA\ULBRA SEMESTRES\Ulbra 2025-01\Compiladores\Trabalho Final - Criação de um Analisador Léxico e Sintático - Ferramentas de Análise\Etapa 3 - Final - Cópia>java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalisadorSintatico
Iniciando análise sintática...
Reconhecido: WHILE
Código válido.

Exemplo de código inválido:



```
1 int y = 0;
2
3 while (y == 10) {
4     y = y + 1;
5 }
6
```

PROBLEMAS 1 SAÍDA CONSOLE DE DEPURAÇÃO **TERMINAL** PORTAS

Couldn't repair and continue parse at character 3 of input
Exception in thread "main" java.lang.Exception: Can't recover from previous error(s)
at java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:392)
at java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:539)
at java_cup.runtime.lr_parser.parse(lr_parser.java:731)
at AnalisadorSintatico.main(AnalisadorSintatico.java:300)



```
1 int i;
2
3 for (i = 0; i < 10; i = i + 1) {
4     i = i + 1;
5 }
6
```

PROBLEMAS 1 SAÍDA CONSOLE DE DEPURAÇÃO **TERMINAL** PORTAS

Couldn't repair and continue parse at character 3 of input
Exception in thread "main" java.lang.Exception: Can't recover from previous error(s)
at java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:392)
at java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:539)
at java_cup.runtime.lr_parser.parse(lr_parser.java:731)
at AnalisadorSintatico.main(AnalisadorSintatico.java:300)

Como Compilar e Executar o Projeto

Pré-requisitos

- Java JDK instalado (recomendado: versão 8 ou superior)
- Terminal ou prompt de comando (cmd, PowerShell, etc.)
- Arquivos java-cup-11b.jar e jflex-full-1.8.1.jar disponíveis na mesma pasta do projeto

Etapa 1 – Gerar o Analisador Léxico com JFlex

```
java -jar jflex-full-1.8.1.jar AnalisadorLexico.flex
```

OBS: após essa etapa, é necessário inserir no AnalisadorLexico.java o import `java_cup.runtime.Symbol`; na primeira linha, conforme imagem abaixo.

```
1 // DO NOT EDIT
2 // Generated by JFlex 1.8.1 http://jflex.de/
3 // source: AnalisadorLexico.flex
4 import java_cup.runtime.Symbol;
5
```

Etapa 2 – Gerar o Analisador Sintático com CUP

```
java -jar java-cup-11b.jar -parser AnalisadorSintatico -symbols sym
AnalisadorSintatico.cup
```

Isso vai gerar os seguintes arquivos:

- AnalisadorSintatico.java
- sym.java

Etapa 3 – Criar o arquivo de teste

Crie um arquivo teste.txt com código a ser testado, por exemplo:

```
int x = 5, y;

if (x > 2) {

    y = x + 1;

} else {

    y = x - 1;}


```

Etapa 4 – Compilar os Arquivos Java

```
javac -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" *.java
```

Etapa 5 – Rodar o Analisador Léxico

```
java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalisadorSintatico
```

Com isso o retorno deve ser esse

Código válido.

Quando não for aceito, sendo invalido a mensagem será algo parecido a isso:

java -cp ".;java-cup-11b.jar;java-cup-11b-runtime.jar" AnalisadorSintatico

Iniciando análise sintática...

Syntax error at character 3 of input

instead expected token classes are [MENOS, MULT, DIV]

Couldn't repair and continue parse at character 3 of input

Exception in thread "main" java.lang.Exception: Can't recover from previous error(s)

at java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:392)

at java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:539)

at java_cup.runtime.lr_parser.parse(lr_parser.java:731)

at AnalisadorSintatico.main(AnalisadorSintatico.java:300)