Gramática definida baseada na linguagem Fortall - Rafael Carneiro Pregardier

```
grammar fortall;

programa: 'programa' ID ';' declaracao* funcaoPrincipal funcao* EOF;

funcaoPrincipal: 'retorna' 'nada' 'funcao' 'principal' '(' ')' bloco;

declaracao: declaracaoVariavel | declaracaoFuncao;

declaracaoVariavel: tipo ID ('=' expressao)? ';';

declaracaoFuncao: 'retorna' (tipo | 'nada') 'funcao' ID '(' parametros? ')' bloco;

funcao: declaracaoFuncao;

parametros: tipo ID (',' tipo ID)*;

tipo: 'int' | 'bool';

// Comandos
bloco: '{' comando* '}';

comando:
    declaracaoVariavel |
    atribuicao |
    chamadaFuncao ';' |
    comandoSe |
    comandoEnquanto |
    comandoEscreva |
    comandoLeia |
    comandoRetorna |
    bloco;

atribuicao: ID '=' expressao ';';

comandoSe: 'se' '(' expressao ')' comando ('senao' comando)?;

comandoEnquanto: 'enquanto' '(' expressao ')' comando;

comandoEscreva: 'escreva' '(' listaExpressoes ')' ';';

comandoLeia: 'leia' '(' ID ')' ';';

comandoRetorna: 'retorna' expressao? ';';

listaExpressoes: expressao (',' expressao)*;
```

```
// Expressões com precedência
expressao: expressaoOu;

expressaoOu: expressaoE ('||' expressaoE)*;

expressaoE: expressaoIgualdade ('&&' expressaoIgualdade)*;

expressaoIgualdade: expressaoRelacional (('==' | '!=') expressaoRelacional)*;

expressaoRelacional: expressaoAditiva (('<' | '>' | '<=' | '>=') expressaoAditiva)*;

expressaoAditiva: expressaoMultiplicativa (('+' | '-') expressaoMultiplicativa)*;

expressaoMultiplicativa: expressaoUnaria (('*' | '/' | '%') expressaoUnaria)*;

expressaoUnaria: ('!' | '-' | '+')? expressaoPrimaria;

expressaoPrimaria:
    NUMERO |
    BOOLEAN |
    STRING |
    ID |
    chamadaFuncao |
    '(' expressao ')';

chamadaFuncao: ID '(' argumentos? ')';

argumentos: expressao (',' expressao)*;

ID: [a-zA-Z_][a-zA-Z0-9_]*;
NUMERO: [0-9]+;
BOOLEAN: 'true' | 'false';
STRING: '"' (~["\\\r\n] | '\\n' | '\\t' | '\\\\' | '\\"')* '"';

COMENTARIO: '/*' .*? '*/' -> skip;
WS: [ \t\r\n]+ -> skip;
```

SIMULADOR DA LINGUAGEM FORTALL - INSTALAÇÃO E DOCUMENTAÇÃO TÉCNICA

RAFAEL CARNEIRO PREGARDIER

Índice

---

1. Requisitos do Sistema

Sistema Operacional:

- Windows 10/11, macOS ou Linux
- Mínimo 4GB RAM
- 500MB espaço em disco

Software Necessário:

- Python 3.8 ou superior
- Java 8 ou superior (para ANTLR)
- Editor de texto ou IDE

---

2. Instalação das Ferramentas

PASSO 1: Instalar Python

1. Baixar Python em: https://www.python.org/downloads/
2. Durante a instalação, marcar "Add Python to PATH"
3. Verificar instalação:python --version

   pip --version

PASSO 2: Instalar Java

1. Baixar OpenJDK em: https://adoptium.net/
2. Instalar e configurar JAVA_HOME
3. Verificar instalação:java -version

PASSO 3: Instalar ANTLR4

1. Baixar ANTLR JAR:wget https://www.antlr.org/download/antlr-4.13.2-complete.jar
2. Criar script/alias para ANTLR:
    - Windows (antlr.bat):java -jar antlr-4.13.2-complete.jar %*
    - Linux/Mac (~/.bashrc):alias antlr='java -jar /path/to/antlr-4.13.2-complete.jar'

PASSO 4: Instalar ANTLR4 Python Runtime

pip install antlr4-python3-runtime==4.13.2

---

3. Configuração do Ambiente

Estrutura de Diretórios

Fortall-Language-Simulator/

```
├── antlr/                  # Gramática ANTLR
│   └── fortall.g4          # Definição da gramática
├── examples/               # Exemplos de código Fortall
│   ├── exemplo1.txt
│   ├── exemplo2.txt
│   ├── exemplo3.txt
│   ├── exemplo4.txt
│   └── exemplo5.txt
├── src/                    # Documentação adicional
│   └── Gramatica_Fortall.txt
├── main.py                 # Programa principal
├── fortallSemantic.py      # Analisador semântico
```

├── fortallInterpreter.py # Interpretador

├── fortall*.py # Arquivos gerados pelo ANTLR

└── INSTALACAO_E_DOCUMENTACAO.txt

Gerar Código Python do ANTLR

1. Navegar até o diretório do projeto
2. Executar comando:antlr -Dlanguage=Python3 -visitor -o . antlr/fortall.g4

Isso gera os arquivos:

- fortallLexer.py
- fortallParser.py
- fortallVisitor.py
- fortallBaseVisitor.py

---

4. Estrutura do Projeto

Componentes Principais

1. **Analisador Léxico (fortallLexer.py)**
    - Gerado automaticamente pelo ANTLR
    - Reconhece tokens: palavras-chave, identificadores, números, strings
    - Ignora comentários e espaços em branco
2. **Analisador Sintático (fortallParser.py)**
    - Gerado automaticamente pelo ANTLR
    - Constrói árvore sintática abstrata (AST)
    - Implementa gramática livre de contexto
3. **Analisador Semântico (fortallSemantic.py)**
    - Implementado manualmente usando padrão Visitor
    - Verifica tipos, declarações, escopo
    - Detecta erros semânticos
4. **Interpretador (fortallInterpreter.py)**
    - Implementado manualmente usando padrão Visitor
    - Executa código em tempo real
    - Gerencia memória e estado do programa
5. **Interface Principal (main.py)**
    - Menu interativo para seleção de exemplos
    - Coordena todas as fases de análise
    - Carrega exemplos de arquivos externos

---

5. Implementação dos Analisadores

Analisador Léxico

- **Abordagem:** ANTLR4 Generator
    - Gramática definida em `fortall.g4`
    - Tokens regulares definidos por expressões regulares
    - Automata finito gerado automaticamente
- **Tokens reconhecidos:**
    - Palavras-chave: `programa`, `retorna`, `funcao`, `se`, `enquanto`, etc.
    - Identificadores: `[a-zA-Z_][a-zA-Z0-9_]*`
    - Números: `[0-9]+`
    - Booleanos: `true`, `false`
    - Strings: com suporte a escape sequences
    - Operadores: aritméticos, lógicos, relacionais
    - Delimitadores: parênteses, chaves, ponto-e-vírgula

Analisador Sintático

- **Abordagem:** ANTLR4 LL(*) Parser
    - Gramática LL(*) com lookahead adaptativo
    - Eliminação de recursão à esquerda automática
    - Precedência de operadores definida na gramática
    - Árvore sintática construída automaticamente
    - Tratamento de erros integrado
- **Características:**
    - Parser top-down recursivo descendente
    - Backtracking automático quando necessário
    - Recuperação de erros com mensagens descritivas
    - Geração de AST navegável

Analisador Semântico

- **Abordagem:** Visitor Pattern Manual
    - Implementa `fortallVisitor` gerado pelo ANTLR
    - Percorre AST em ordem depth-first
    - Mantém contexto semântico durante análise
- **Funcionalidades implementadas:**
    a. **Verificação de Tipos**
        - Sistema de tipos forte (`int`, `bool`, `string`)
        - Verificação de compatibilidade em atribuições
        - Verificação de tipos em expressões
        - Coerção automática limitada
    b. **Análise de Escopo**
        - Escopo global e local (funções)
        - Tabela de símbolos hierárquica
        - Verificação de declarações duplicadas
        - Verificação de uso antes da declaração

    c. **Análise de Funções**
        - Verificação de assinatura de funções
        - Compatibilidade de argumentos
        - Verificação de retorno obrigatório
        - Análise de recursão
    d. **Verificação de Fluxo**
        - Inicialização de variáveis
        - Comandos de retorno em contexto correto
        - Condições válidas em estruturas de controle

Interpretador

- **Abordagem:** Tree-Walking Interpreter
    - Execução direta sobre a AST
    - Padrão Visitor para navegação
    - Gerenciamento de estado em tempo real
- **Características de implementação:**
    a. **Gerenciamento de Memória**
        - Variáveis globais e locais separadas
        - Pilha de chamadas para funções
        - Limpeza automática de escopo
    b. **Execução de Expressões**
        - Avaliação bottom-up
        - Respeita precedência de operadores
        - Operações aritméticas, lógicas e relacionais
    c. **Controle de Fluxo**
        - Implementação de `se`/`senao`
        - Loops `enquanto` com condição
        - Comando `retorna` com controle de fluxo
    d. **Sistema de Funções**
        - Chamadas de função com parâmetros
        - Valores de retorno
        - Recursão suportada
        - Escopo local isolado
    e. **Entrada/Saída**
        - Comando `escreva` para saída
        - Comando `leia` para entrada do usuário
        - Conversão automática

6. Arquitetura do Sistema

Fluxo de execução

O processo de execução do simulador da linguagem Fortall segue uma sequência de fases:

1. **Carregamento**
    - Carrega o código fonte de um arquivo `.txt`.

    - Cria um `InputStream` para o ANTLR.
    - Inicializa os componentes necessários.
2. **Análise léxica**
    - O `fortallLexer` processa os caracteres do código fonte.
    - Gera um `stream` de tokens.
    - Filtra comentários e espaços em branco.
3. **Análise sintática**
    - O `fortallParser` consome os tokens gerados.
    - Constrói a Árvore Sintática Abstrata (AST).
    - Reporta quaisquer erros sintáticos encontrados.
4. **Análise semântica**
    - O `SemanticAnalyzer` visita a AST.
    - Verifica a compatibilidade de tipos e o escopo das variáveis.
    - Constrói a tabela de símbolos do programa.
    - Reporta quaisquer erros semânticos detectados.
5. **Interpretação (se sem erros)**
    - O `FortallInterpreter` visita a AST.
    - Executa os comandos do programa em ordem.
    - Gerencia o estado do programa em tempo real.
    - Produz a saída conforme a execução do código.

7. Como Executar

Execução básica

Para executar os exemplos predefinidos do simulador:

1. Navegue até o diretório raiz do projeto:cd Fortall-Language-Simulator
2. Execute o programa principal:python main.py
3. Siga as instruções do menu interativo no console:
    - Escolha um exemplo (de 1 a 5).
    - Visualize o código fonte do exemplo selecionado.
    - Confirme a execução.
    - Interaja com o programa, se ele solicitar entrada do usuário.

Execução personalizada

Para executar seu próprio código Fortall:

1. Crie um novo arquivo `.txt` dentro da pasta `examples/`.
2. Escreva seu código Fortall, garantindo que ele siga a sintaxe da linguagem.
3. Adicione uma nova entrada para o seu arquivo no dicionário `examples` dentro do arquivo `main.py`.
4. Execute o programa normalmente, seguindo os passos da "Execução Básica".

8. Exemplos disponíveis

Catálogo de exemplos

O simulador inclui os seguintes exemplos para demonstração:

1. **EXEMPLO 1 - Básico (exemplo1.txt)**
    - Contém apenas a função principal.
    - Demonstra operações aritméticas simples.
    - Inclui uma estrutura condicional básica.
    - Tem como objetivo apresentar os conceitos fundamentais da linguagem.
2. **EXEMPLO 2 - Funções (exemplo2.txt)**
    - Apresenta uma função principal e uma função auxiliar.
    - Ilustra a passagem de parâmetros entre funções.
    - Mostra como utilizar valores de retorno de funções.
    - Inclui uma estrutura de repetição.
3. **EXEMPLO 3 - Recursão (exemplo3.txt)**
    - Implementa o cálculo de Fibonacci de forma recursiva.
    - Demonstra o uso de recursão profunda.
    - Envolve controle de fluxo mais complexo.
    - Pode abordar otimização de chamadas (se implementado na linguagem).
4. **EXEMPLO 4 - Calculadora Interativa (exemplo4.txt)**
    - Utiliza múltiplas funções para organizar a lógica.
    - Permite entrada de dados do usuário através do comando `leia`.
    - Realiza operações matemáticas diversas.
    - Pode incluir verificação de propriedades ou condições.
5. **EXEMPLO 5 - Jogo Interativo (exemplo5.txt)**
    - Contém uma lógica de jogo simples.
    - Integra múltiplas funções que colaboram entre si.
    - Apresenta uma interface de usuário mais avançada (dentro das capacidades da linguagem).
    - Gerencia um estado complexo do programa.