Documentação Técnica – Compilador PORTUGOL

Integrantes do Grupo

- Julia Rampin – 2205949
- Bruno Vaz – 2202162
- Pietro Ramos Victorino – 2203760
- Gabriel Reis Quintiliano – 2201174

Visão Geral

Este projeto implementa um compilador didático para a linguagem PORTUGOL, estruturado em três fases principais: análise léxica, análise sintática e análise semântica. Cada fase é representada por um módulo específico. A coordenação de execução é realizada no arquivo `main.py`.

Estrutura do Projeto

- `main.py`: Coordenador de execução.
- `analisador_lexico.py`: Analisador léxico (Fase 2).
- `analisador_sintatico.py`: Analisador sintático (Fase 3).
- `analisador_semantico.py`: Analisador semântico (Fase 4).
- `exemplo.POR`: Arquivo de entrada com código-fonte PORTUGOL.
- `saida.TEM`: Arquivo gerado com a lista de tokens.
- `saida.csv`: Arquivo gerado para visualização tabular dos tokens.

---

1. `main.py` – Executor

Executa, em sequência, as três fases do compilador:

1. Chama `analisador_lexico` com entrada `.POR` e saída `.TEM`.
2. Invoca `Parser` para processar tokens sintaticamente.
3. Executa `AnalisadorSemantico` para validação de uso de variáveis.
4. Gera arquivo CSV a partir de `saida.TEM`.

---

2. `analisador_lexico.py` – Análise Léxica

Objetivo

Converter o código-fonte em uma sequência de tokens válidos, armazenando-os em `saida.TEM`.

Funcionalidade

- Usa expressões regulares para identificar tokens:
  - Palavras reservadas (e.g., `se`, `para`, `fim_se`)
  - Operadores (`+`, `-`, `*`, `/`, `<-`, etc.)
  - Delimitadores (`(`, `)`, `:`, `;`)
  - Literais (`"texto"`, `10`)
  - Identificadores (variáveis)
- Reconhece declarações no formato: `inteiro:variavel inteiro variavel`
- Monta uma tabela de símbolos associando cada identificador (`ID`) a uma posição numérica.
- Gera o arquivo `saida.TEM` no formato: `TIPO - ID 0 ATR - NUMINT -`

---

3. `analisador_sintatico.py` – Análise Sintática

Objetivo

Validar a estrutura gramatical dos tokens segundo a linguagem PORTUGOL.

Funcionalidade

- Lê `saida.TEM` e aplica regras da gramática.
- Estruturas suportadas:
  - Declarações: `TIPO [DOISPONTOS] ID [;]`
  - Atribuições: `ID <- EXPRESSAO [;]`
  - Entrada/Saída: `leia(ID)`, `escreva(ID ou STRING)`
  - Condicional: `se EXPRESSAO_LOGICA então ... fim_se`
  - Repetição: `para ID <- NUM até NUM [passo NUM] ... fim_para`
- O ponto e vírgula `;` é opcional:
  - Se ausente, apenas um aviso (`print`) é gerado, não impedindo a execução.

Relatórios de erro:

-
  - Quando tokens estão fora de ordem, faltando, ou em locais inesperados.

---

4. `analisador_semantico.py` – Análise Semântica

Objetivo

Validar o uso correto de variáveis no código, com base nas declarações anteriores.

Funcionalidade

- Rastreia identificadores declarados na tabela de símbolos.
- Marca variáveis como declaradas ao encontrar `TIPO DOISPONTOS ID`.
- Verifica cada uso posterior (`ID`) e reporta erro se não declarado: `Variável usada sem declaração prévia: posição X`
- Suporta verificações em:
  - Expressões aritméticas
  - Leitura (`leia`)
  - Escrita (`escreva`)
  - Condicionais e laços (`se`, `para`)

---

5. Formatos de Arquivos

5.1 Entrada `.POR`

Arquivo contendo código-fonte PORTUGOL, como:

```
portugol inteiro:idade idade <- 20 escreva("Idade:")
```

5.2 Tokens `.TEM`

Gerado pelo analisador léxico. Exemplo:

```
TIPO - DOISPONTOS - ID 0 ATR - NUMINT - ESCREVA - PARAB - STRING
- PARFE - EOF -
```

5.3 Tokens `.CSV`

Usado para visualização em planilha:

| Token | Posição | |-----------|---------| | TIPO | - | | ID | 0 | | ATR | - |

---

6. Observações

- Variáveis são mapeadas por posição numérica, e validadas por chave.
- O sistema não executa o código, apenas valida sintaticamente e semanticamente.
- O compilador aceita ausência de `;`, sendo tolerante, porém sinaliza por aviso.

---

7. Requisitos

- Python 3.6+
- Arquivo de entrada `.POR` codificado em UTF-8

Desenvolvido como atividade acadêmica para fins de aprendizado sobre construção de compiladores.