Projeto de Compiladores

Autor: Bruno Zabot - 202302069

9 de Novembro de 2025

Conteúdo

1 Introdução

Este projecto implementa uma Arvore Sintática Abstrata de compiladores, que interpreta e executa uma sub-linguagem baseada na linguagem Ada. O objectivo principal é realizar a análise, interpretação e simulação da execução de pequenos programas escritos nesta linguagem.

2 Como Funciona

O projecto segue um processo dividido em três fases principais:

- **Análise e Tokenização:** O código-fonte é analisado lexicalmente e convertido em *tokens*. Um analisador sintático (parser) converte esses *tokens* numa Árvore de Sintaxe Abstracta (AST).
- **Árvore Sintática Abstracta (AST):** A AST é uma estrutura em árvore que representa a lógica do programa. Suporta expressões (ex: operacões aritméticas e booleanas) e comandos (como atribuições e controlo de fluxo).
- **Interpretador:** O interpretador executa os comandos da AST, avaliando expressões e exibindo os resultados.

3 Arquitetura e Implementação

O processo de interpretação do código-fonte segue um fluxo bem definido, desde a leitura do ficheiro até à execução final.

1. O utilizador fornece um ficheiro de código-fonte com sintaxe Ada-like como entrada.
2. O código é processado pelo analisador lexical (`scanner.flex`), que o divide em *tokens*. Estes tokens são então enviados para o analisador sintático (`parser.bison`).
3. O parser, com base nas regras gramaticais definidas, constrói uma Árvore Sintatica Abstrata (AST). A estrutura da AST é definida em `ast.h`.
4. O ficheiro `interpreter.c` orquestra este processo e, após a construção da AST, invoca as funções de impressão, `printAST`.

Componentes Principais

- `scanner.flex`: Responsável pela análise lexical. É configurado para ser *case-insensitive* e reconhece palavras reservadas, números (inteiros e *float*) e operadores.
- `parser.bison`: Contém a gramática da linguagem. Define os tokens, as precedências de operadores e as regras para a construção da AST. Suporta expressões simples, comandos e programas completos na estrutura Ada.
- `ast.h`: Define as estruturas de dados para a AST, incluindo nós para programas, comandos (`if`, `while`), expressões aritméticas e booleanas.
- `interpreter.c`: O ponto de entrada que chama o `yyparse` para iniciar a análise e, subsequentemente, percorre a AST para interpretar o código.

4 Limitações

- Não suporta comentários.

5 Executar os Testes

Para verificar rapidamente o funcionamento com os exemplos fornecidos, execute o seguinte comando no directório:

```
make test
```

Este comando irá compilar o projecto e executar o interpretador com os ficheiros de exemplo localizados no directório `examples`. Ou então, caso queira rodar com um caso em específico:

```
make && ./interpreter < {your_example}
```