Relatório - Analisador Léxico e Sintático para PL/0

Identificação

- **Disciplina:** SCC0605 Teoria da Computação e Compiladores
- **Instituição:** ICMC/USP
- **Professor:** Thiago A. S. Pardo
- **Trabalho 2:** Análise Sintática
- **Grupo:**
    - Leonardo Gueno Rissetto NUSP: 13676482
    - Lucas Lima Romero NUSP: 13676325
    - Luciano Gonçalves Lopes Filho NUSP: 13676520
    - Marco Garcia NUSP: 11833581
    - Thiago Kashivagui Gonçalves NUSP: 13676579

---

1. Introdução

Este projeto implementa um analisador léxico e sintático para a linguagem PL/0, integrando conceitos fundamentais de compiladores estudados na disciplina SCC0605. O sistema foi desenvolvido em C, com foco em modularidade, robustez e facilidade de uso. O analisador léxico identifica e classifica os elementos básicos do código-fonte, enquanto o analisador sintático verifica a estrutura do programa conforme a gramática da linguagem, reportando todos os erros encontrados. O projeto enfatiza o tratamento eficiente de erros e a clareza das mensagens ao usuário, servindo como base sólida para etapas futuras do desenvolvimento de compiladores.

---

2. Correções realizadas a partir do Trabalho 1

- **Detecção de erro em comentários:**
    - Corrigido o analisador léxico para detectar corretamente erros em comentários, conforme a especificação da linguagem PL/0, que permite apenas comentários de uma linha. Antes, o analisador permitia comentários em múltiplas linhas, o que não estava de acordo com a gramática. Agora, qualquer comentário não fechado corretamente na mesma linha é identificado como erro léxico.
- **Detecção de números mal formados:**

- Corrigido um problema na detecção de números mal formados no analisador léxico, onde um caractere a mais era pulado após a identificação do erro. Agora, o analisador consome corretamente apenas os caracteres pertencentes ao número mal formado, garantindo precisão na análise e na mensagem de erro.

---

3. Decisões de Projeto

- **Léxico:**
  - Implementado em C, com foco em modularidade e eficiência.
  - Utiliza tabelas hash para reconhecimento rápido de palavras reservadas e símbolos.
  - Reconhece identificadores, números, palavras reservadas, operadores, símbolos e comentários, além de tratar erros como números mal formados, comentários não fechados e caracteres inválidos.
  - O tratamento de erros léxicos é imediato: ao encontrar um erro, o token de erro é emitido e a análise prossegue, permitindo a detecção de múltiplos problemas em uma única execução.
- **Sintático:**
  - Implementado como um analisador descendente preditivo recursivo, com uma função para cada não-terminal da gramática PL/0.
  - O tratamento de erros sintáticos é feito pelo modo pânico, utilizando conjuntos de sincronização específicos para cada produção, permitindo recuperação e continuidade da análise após erros.
  - O parser cobre todas as construções da linguagem PL/0: declarações de constantes, variáveis, procedimentos, comandos compostos, comandos de atribuição, condicionais, laços, expressões aritméticas e relacionais, respeitando a precedência dos operadores.
  - Relata múltiplos erros sintáticos em uma única execução, sem abortar na primeira falha.
- **Integração:**
  - O analisador sintático consome tokens do léxico de forma transparente, por meio da função `advance()`.
  - Todos os erros léxicos e sintáticos são reportados em um arquivo de saída, com informações detalhadas (linha, tipo de erro, token).
  - Ao final da análise, se não houver erros, uma mensagem de sucesso é registrada.
- **Modularidade e Organização:**
  - O código está dividido em módulos bem definidos:
    - `lexico.*`: analisador léxico e funções auxiliares
    - `sintatico.*`: analisador sintático e controle de parsing
    - `hash_table.*`: implementação das tabelas hash para palavras e símbolos reservados
    - `main.c`: ponto de entrada, integração e controle de arquivos
  - O uso de cabeçalhos (`.h`) garante encapsulamento e facilita manutenção e testes.
  - O projeto segue boas práticas de documentação interna e uso de comentários explicativos.

---

4. Estratégias de Tratamento de Erros, Usabilidade e Limitações

- **Tratamento de erros léxicos e sintáticos:**
    - O modo pânico está implementado no parser, permitindo que a análise prossiga após erros e relatando todos os problemas encontrados.
    - O léxico emite tokens de erro para qualquer situação inesperada, e o parser trata esses tokens de forma transparente.
    - O sistema foi projetado para ser tolerante a erros, maximizando a utilidade para o usuário e facilitando a depuração de programas PL/0.
- **Mensagens e usabilidade:**
    - Todas as mensagens de erro são detalhadas, informando a linha, o tipo de erro e o token envolvido.
    - O arquivo de saída é legível e pode ser usado diretamente pelo usuário para corrigir o código-fonte.
    - A interface Python permite uso simples, sem necessidade de interação manual com arquivos de entrada/saída.
- **Extensibilidade e manutenção:**
    - O uso de tabelas hash para palavras e símbolos reservados facilita a inclusão de novos elementos na linguagem.
    - O código modular e documentado permite fácil adaptação para futuras etapas do compilador (ex: análise semântica ou geração de código).
    - O projeto foi estruturado para facilitar testes unitários e integração contínua.
- **Limitações e pontos de atenção:**
    - O analisador não realiza análise semântica (ex: verificação de tipos ou declaração prévia de identificadores), pois isso não faz parte do escopo deste trabalho.
    - O sistema espera que o arquivo de entrada esteja no formato correto (UTF-8, texto puro).
    - Recomenda-se sempre revisar o arquivo de saída após a execução para garantir que não houve erros não tratados.

---

5. Instruções para Compilar e Executar

Requisitos

- Compilador C (testado com `gcc`)
- Python 3 (para interface)
- Sistema operacional: Linux, macOS ou Windows

Execução via interface Python

O modo recomendado de uso é pela interface Python fornecida no arquivo `interface_tester.py` na raiz do projeto. Ela automatiza a execução do analisador e facilita a interação com o usuário.

No terminal, execute:

```
python3 interface_tester.py
```

Na interface Python, escreva o código PL/0 na caixa destinada a isso e clique para executar a análise. O resultado da execução, incluindo as linhas com erro (caso existam), será exibido diretamente na própria interface. Se não houver erros, a mensagem de sucesso será mostrada na interface.

Execução manual (opcional)

O programa espera um arquivo de entrada em `input/codigo.pl0` e gera a saída em `output/saida_sintatico.txt`.

No terminal, dentro da pasta `Código/` execute:

```
make run
```

Parâmetros

- **Entrada:** `input/codigo.pl0` (programa em PL/0)
- **Saída:** `output/saida_sintatico.txt` (relatório de erros ou sucesso)

---

6. Exemplo de Execução

Exemplo sem erro:

```
CONST a = 10, b = 20;
VAR x, y;
BEGIN
  x := a + b;
  y := x * 2
END.
```

Saída:

```
Nenhum erro encontrado.
```

Exemplo com erro:

```
VAR n, fat;
BEGIN
        n:=4;
        fat:=1;
        WHILE n > 1 DO
                BEGIN
                fat:=fat n;
                n:=n-1;
                END
END.
```

Saída:

```
Erro sintático na linha 7: Esperado operador aritmético (+, -, * ou /). Token atual: n
```

---

7. Conclusão

O desenvolvimento deste projeto proporcionou uma compreensão prática dos conceitos de análise léxica e sintática, além de reforçar a importância da modularidade, tratamento de erros e clareza na comunicação com o usuário. A implementação do modo pânico tornou o sistema mais robusto, permitindo identificar múltiplos erros em uma única execução e facilitando o uso para estudantes e professores.

Como grupo, destacamos o valor do trabalho colaborativo e da documentação clara, que foram essenciais para superar desafios técnicos e garantir a qualidade do código. Acreditamos que o projeto está bem estruturado para servir de base para etapas futuras do compilador, como análise semântica e geração de código.
