UNIVERCIDADE AGOSTINHO NETO

FACULDADE DE CIÊNCIAS NATURAIS

DEPARTAMENTO DE COMPUTAÇÃO

TRABALHO COLETIVO DE PROGRAMAÇÃO IMPERATIVA

PROJECTO : "**ANALISADOR DE EXPRESSÕES MATEMÁTICAS"**

GRUPO N° 21

Por:

Amândio de Jesus Almada

Luanda 2024/2025

INTEGRANTES DO GRUPO

1- João José Zinguila Mário
2- Mena Miguel Tambilo
3- Milzia Arieth Tchipepe Java
4- Ponteciana António Gabriel

Índice

Introdução

Este relatório apresenta o desenvolvimento do projeto "Analisador de Expressões Matemáticas" em linguagem C, realizado no âmbito da disciplina de Programação Imperativa. O objetivo principal foi consolidar conhecimentos em estruturas de dados, modularização e manipulação de cadeias de caracteres em C.

O contexto da implementação parte da necessidade de desenvolver um sistema capaz de interpretar expressões matemáticas simples (como 2 + 3 * (4 - 1)), tokenizá-las e verificar a correção dos parênteses, abrindo caminho para futuras expansões, como a avaliação da expressão ou geração de árvores sintáticas.

O projeto foi escolhido por integrar conceitos fundamentais de programação como:

- Análise léxica;
- Manipulação de vetores e estruturas;
- Modularização do código.

Além de promover o raciocínio lógico, permitiu compreender melhor como funciona a base de um compilador simples.

Descrição do projeto

Objetivos específicos:

Criar um analisador léxico (tokenizador) para expressões matemáticas.

Verificar o balanceamento de parênteses nas expressões.

Validar a estrutura dos tokens gerados.

Modularizar o projeto com cabeçalhos (.h) e implementação (.c).

Funcionalidades principais:

Leitura de uma expressão matemática como entrada.

Tokenização da expressão, classificando números, operadores e parênteses.

Verificação de parênteses balanceados.

Impressão dos tokens e feedback sobre possíveis erros.

Estrutura do projeto:

/projeto-analisador/

```
│
├── main.c              → Função principal.
├── lexer.h / lexer.c   → Tokenização da expressão.
├── parser.h / parser.c → Análise dos tokens (parênteses, estrutura).
├── evaluator.h         → Enumeração de tipos de tokens.
├── evaluator.c         →  Avaliação de expressão.
```

Metodologia e desenvolvimento

❖ Implementação

O desenvolvimento iniciou-se com a criação de um módulo léxico (lexer.c) para dividir a expressão em tokens. A seguir, foi desenvolvido um parser simples para verificar parênteses balanceados, e por fim, o main.c unifica as chamadas e faz os testes.

❖ Algoritmos e estruturas de dados

Estrutura Token { tipo, valor }

Enumeração TokenTipo com categorias: NUMERO, OPERADOR, PARENTESE_ESQ, PARENTESE_DIR.

Uso de isdigit() para identificação de números.

Algoritmo de contagem de parênteses com incremento/decremento e detecção de desequilíbrio.

❖ Modularização

Cada componente foi isolado:

lexer para análise léxica

parser para verificação sintática

tokens.h para padronizar os tipos Essa abordagem facilitou testes, reutilização e manutenção do código.

❖ Dificuldades enfrentadas

Manipulação de caracteres e strings sem string.h avançado exigiu lógica manual para identificar números de mais de um dígito.

Balanceamento de parênteses exigiu atenção à ordem correta.

A modularização em arquivos separados demandou organização cuidadosa nos includes e headers.

Essas dificuldades foram superadas com pesquisas, testes incrementais e apoio de colegas.

Resultados e testes

Funcionalidades implementadas com sucesso

Tokenização correta de expressões simples (com e sem espaços).

Identificação de tipos corretos para cada caractere.

Verificação de parênteses balanceados.

Impressão organizada de tokens com tipo e valor.

O sistema de tokenização foi robusto para várias entradas, mesmo com operadores repetidos ou parênteses encadeados.

Modularização facilitou correções e adição de novas funcionalidades.

Aspectos a melhorar

O sistema ainda não avalia o valor da expressão (isso poderia ser uma extensão futura).

A tokenização não trata corretamente números com mais de um dígito (ex: 23).

Poderia haver mais mensagens de erro detalhadas para auxiliar o usuário.

Conclusão

Este projeto permitiu consolidar os fundamentos da linguagem C, especialmente o uso de estruturas, enums, vetores e funções. A implementação de um analisador léxico e verificador de parênteses mostrou-se um excelente exercício de lógica e modularização.