CIÊNCIAS DA COMPUTAÇÃO

FACULDADE DE CIÊNCIAS NATURAIS | UNIVERSIDADE AGOSTINHO NETO

RELATÓRIO DO PROJETO

PROGRAMAÇÃO IMPERATIVA

TEMA:

ANALISADOR DE EXPRESSÕES MATEMÁTICAS

Grupo nº 3

Feito por:

1- Valdemiro Quental
2- Bruno Francisco
3- Filismino Viana
4- Jório Rosa

Assinatura

Luanda – 2025

INTRODUÇÃO

Este relatório descreve o desenvolvimento de um projeto prático na disciplina de Programação Imperativa, cujo objetivo principal foi criar um analisador de expressões matemáticas utilizando a linguagem de programação C.

O contexto do projeto está relacionado à necessidade de compreender como expressões numéricas são interpretadas e processadas por linguagens de programação. Através desta implementação, foi possível estudar e aplicar conceitos como análise léxica, sintática, avaliação de expressões e tratamento de erros.

O projeto visa também familiarizar os estudantes com o uso de estruturas de dados fundamentais como pilhas, filas e listas ligadas, fundamentais na construção de interpretadores simples.

DESCRIÇÃO DO PROJETO

Objetivos específicos:

- Ler expressões matemáticas de um arquivo de texto.
- Analisar a validade de cada expressão.
- Avaliar as expressões válidas e gerar o resultado.
- Reportar erros de sintaxe ou semântica (ex: parênteses errados, divisão por zero, etc.).

Funcionalidades principais:

- Análise léxica (tokenização das expressões).
- Análise sintática (verificação da estrutura).
- Avaliação matemática usando algoritmo de notação pós-fixa (Shunting Yard).
- Relatório de erros no arquivo de saída.

Estrutura do projeto:

O projeto foi organizado de forma modular, com os seguintes arquivos:

- avaliador.c: leitura de entrada/saída e controle do fluxo.
- posfixa.c: análise sintática (validação da estrutura).
- token.c: análise léxica (divisão da expressão em tokens).
- pilha.h/, pilha_int.h/, fila.h/: estruturas auxiliares fornecidas no repositório.

METODOLOGIA E DESENVOLVIMENTO

Implementação:

O projeto foi desenvolvido em C utilizando o Embarcado Dev-C++ para edição e compilação. Foram seguidas práticas de modularização para separar as responsabilidades de cada parte do código.

Algoritmos e estruturas de dados utilizadas:

- Pilhas para validar parênteses e avaliar a expressão em notação pós-fixa.
- Listas de tokens para representar a expressão de forma estruturada.
- Algoritmo de Shunting Yard para conversão de notação infixa para pós-fixa.
- Análise léxica feita com verificação caractere por caractere.

Estratégia de modularização:

Cada componente do projeto foi separado em um módulo .c e .h correspondente, facilitando testes e manutenção.

Dificuldades encontradas:

Compreensão da lógica por trás da tokenização e avaliação de expressões, adaptação das estruturas de dados fornecidas pelo repositório, e tratamento adequado de erros como divisão por zero e caracteres inválidos. Essas dificuldades foram superadas através de pesquisa, testes incrementais e auxílio com colegas e professores.

RESULTADOS E TESTES

Funcionalidades implementadas com sucesso:

- Leitura de expressões do arquivo in.txt.
- Tokenização correta de expressões simples e compostas.
- Detecção de erros de parênteses e caracteres inválidos.
- Escrita dos resultados ou erros no out.txt.

O que pode ser melhorado:

- A detecção de erros pode ser aprimorada para fornecer mensagens mais específicas.
- Adição de suporte a números com mais de um dígito.
- Otimização na conversão para notação pós-fixa.

CONCLUSÃO

O projeto permitiu aplicar na prática conceitos fundamentais da programação imperativa, especialmente estruturas de dados e modularização.

Foi possível perceber como expressões são analisadas em compiladores e como implementar um pequeno interpretador em C.

Ao final, foi possível entregar um programa funcional, modular e com boa organização.

REPOSITÓRIO DO PROJETO

https://github.com/Vquental/Projeto_PI/tree/main/Grupo%203

https://github.com/Vquental/Projeto_PI.git