Trabalho Prático 1
Estrutura de dados

Elissandro Caetano Júnior

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

ecjunior19@ufmg.br

1. Introdução

O trabalho proposto, com objetivo de recordar fundamentos de desenvolvimento em C/C++ e introduzir conceitos como TADs (Tipos Abstratos de Dados), se trata da conversão e resolução de expressões matemáticas infixas (A + B) e posfixas (A B +). Esses formatos de arquivos são caracterizados por uma sequência de números representando, para cada pixel, os tons de vermelho, azul e verde no caso do PPM, e cinza no outro formato.

Como solução, optou-se pela criação de TADs e funções específicas para ligas com cada um dos tipões de anotação. partir dessas instâncias abstratas, foram desenvolvidas funções separadas e consolidadas para fazer a leitura, conversão e escrita dos resultados. Assim, como a implementação de uma pilha auxiliar de alocação dinâmica, para auxiliar na manipulação das expressões.

2. Método

2.1 Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU, dentro de um ambiente virtual Linux.

2.2 Classe

Como uma prática introdutória, a única “estrutura de dados” utilizada na prática foi uma pilha dinamicamente alocada para aceitar valores diferentes, utilizando *Class Templates*, Em busca de uma maior abstração, todos as funções auxiliares são implementadas de maneira decentralizada no código.

2.3. Funções

Descrição das principais funções utilizadas no trabalho.

Bool checkIfPostfix(string str): Verifica se a expressão é profixa.

Bool checkIfInfix(string str): Verifica se a expressão é infixa

String convertePosfixoInfixo(string str): Converte uma expressão posfixa para infixa.

String converteInfixoPosfixo(string str): Converte uma expressão infixa para profixa.

Double resolvePostfix(string str): Resolve expressões válidas na anotação posfixa.

Double resolveInfixa: Resolve expressões válidas na anotação infixa.

bool isNumber(string str): Verificar se a *string* é um número. Funciona para números com casas decimais.

Int precedencia(string str): Retorna o valor de precedências entres os quatro principais operadores (+, -, *, /);

Bool comparaPrecedencia(string s1; string s2): Compara o valor de precedência entre os diferentes operadores;

Double operaçcao(string s1; string s2, string op): Realiza as quatro operações matemáticas básicas entre dois números;

3. Análise de Complexidade

Observando as principais funções de conversão e resolução das expressões infixa e posfixa, elas terão as seguintes complexidades assintóticas:

- **Complexidade de tempo O(N).** Sendo N o tamanho da *string* da expressão matemática.
- **Complexidade de expaço: O(M)**. Sendo M o tamanho da pilha auxiliar.

4. Estratégias de Robustez

Para garantir robustez e a execução correta do programa diante da entrada do usuário, foram empregadas diversas estratégias de robustez, apresentadas no formato bibliotecas auxiliares e função secundárias de validação. Além, da modularização da code base.

Um ponto fraco da robustez do programa, é o fato de que o arquivo de entrada precisa estar corretamente formatado. Caso, o usuário deseje mais flexibilidade na manipulação dos inputs, pode haver impacto na execução funcional do código.

5. Análise Experimental

Como foi visto na análise de complexidade, grande parte da complexidade de tempo das funções de manipulação possui O(N). Para melhor visualização e análise do comportamento assintótico, foi desenvolvido um gráfico demonstrativo.

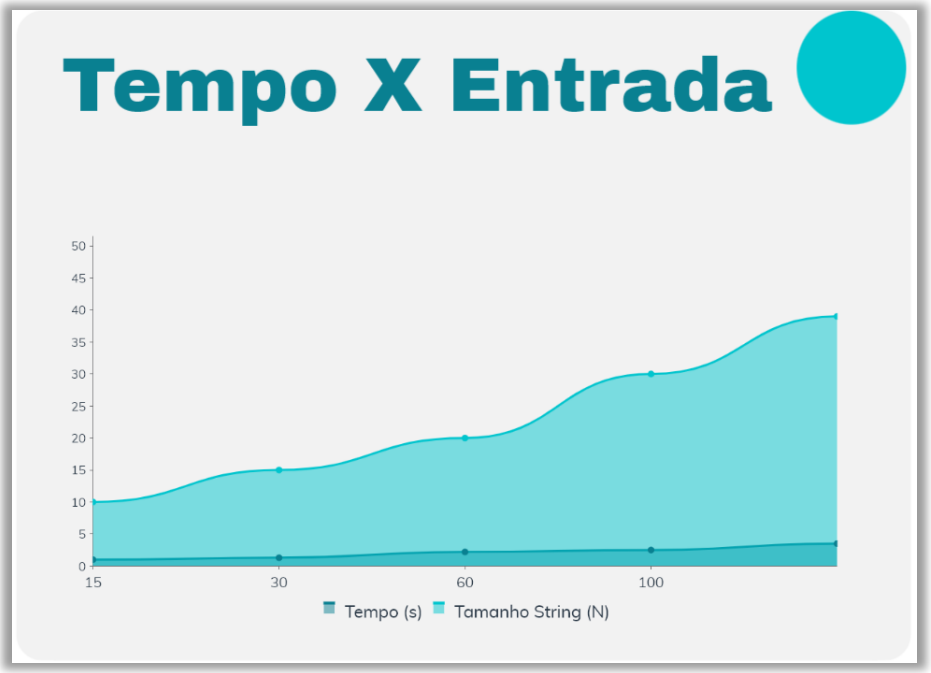

Figura 1 - Gráfico Tempo (s) x Entrada (N)

6. Conclusões

Em conclusão, a implementação da pilha neste trabalho de estrutura de dados foi uma experiência enriquecedora que me permitiu compreender a importância e a utilidade dos TADs. Ao longo do projeto, fui capaze de explorar os conceitos fundamentais da pilha, como sua organização baseada em LIFO (Last-In-First-Out), e aplicá-los na resolução de problemas específicos.

Durante a implementação, enfretei desafios interessantes, como a gestão correta das operações de inserção (push) e remoção (pop) na pilha, garantindo a exatidão e a eficiência das operações. Além disso, aprendia a lidar com casos de pilhas vazias e cheias, implementando mecanismos de verificação e tratamento de erros.

Uma das principais vantagens da utilização da pilha foi sua simplicidade de implementação e sua eficiência em termos de tempo e espaço. Através do uso de uma lista encadeada e o uso de *Class Templates*, pude criar uma estrutura de pilha flexível e adaptável às necessidades do problema em questão.

Além disso, durante o desenvolvimento do projeto, aprimorei minhas habilidades de programação, bem como nossa capacidade de projetar e implementar estruturas de dados eficientes e modularizadas. A experiência também nos proporcionou uma compreensão mais profunda dos princípios de encapsulamento, abstração e reutilização de código.

Em suma, a implementação da pilha neste trabalho me proporcionou uma sólida compreensão dos conceitos fundamentais dessa estrutura de dados e do uso de TADS no geral. Além disso, pude aplicar esse conhecimento na solução de problemas práticos, sendo capaz de testar e aprimorar minhas habilidades.

7. Bibliografia

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

GEEK FOR GEEKS. **Expression Evaluation**. Disponível em: https://www.geeksforgeeks.org/expression-evaluation/. Acesso em: 7 mai. 2023.

GEEK FOR GEEKS. **Postfix to Infix**. Disponível em: https://www.geeksforgeeks.org/postfix-to-infix/#_=_. Acesso em: 5 mai. 2023.

GEEKS FOR GEEKS. **Convert Infix expression to Postfix expression**. Disponível em: https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/. Acesso em: 2 mai. 2023.

GEEKS FOR GEEKS. **Evaluation of Postfix Expression**. Disponível em: https://www.geeksforgeeks.org/evaluation-of-postfix-expression/. Acesso em: 3 mai. 2023.

IME USP. **2.9. Infix, Prefix and Postfix Expressions**. Disponível em: https://panda.ime.usp.br/panda/static/pythonds_pt/02-EDBasicos/InfixPrefixandPostfixExpressions.html. Acesso em: 1 mai. 2023.

JAVA POINT. **Program to convert infix to postfix expression in C++ using the Stack Data Structure**. Disponível em: https://www.javatpoint.com/program-to-convert-infix-to-postfix-expression-in-cpp-using-the-stack-data-structure. Acesso em: 26 abr. 2023.

TUTORIALS POINT. **Postfix to Infix in C++**. Disponível em: https://www.tutorialspoint.com/postfix-to-infix-in-cplusplus. Acesso em: 4 mai. 2023.

8. Instruções para compilação e execução

Para compilar e usar o programa, descompacte o arquivo .zip e acesse o diretório “TP”, considerado o diretório raiz do programa, com seu terminal de preferência. Logo em seguida, utilize o comando “make run” para compilar o código pela primeira vez. Caso, seja preciso o comando “make clean” limpa todos os arquivos criados.

Para executar o programa, acesse o executável gerado com o comando “./bin/main” fornecendo um arquivo de entrada, tendo em vista o diretório em que se encontra o arquivo. Alguns exemplos de uso:

- /bin/main.a < entdouble.s11.n10.i.in
- ./bin/main < entdouble.s4.n5.p.in
- ./bin/main < entdouble.s3.n5.i.in