Implementação e Análise das Estruturas de dados Pilha e Fila

Lucas Fontes Buzuti
Departamento de Engenharia Elétrica
Centro Universitário FEI
São Bernardo do Campo-SP, Brasil
lucas.buzuti@outlook.com

***Resumo*—Esse artigo tem uma finalidade acadêmica na compreensão e implementação das estruturas de dados pilha e fila. Uma análise entre as duas estruturas é proposta. A implementação foi utilizar a programação orientada a objetos na linguagem *C++*, tendo em foco a otimização e a velocidade na execução de algoritmos.**

***Index Terms*—estrutura de dados, pilha, fila, C++**

I. Introdução

Esse artigo tem em seu objetivo a compreensão e implementação das estruturas de dados pilha e fila. Além da compreensão uma análise crítica entre as duas estruturas é proposta. A implementação tem como alvo a utilização da linguagem *C++*, pois é uma linguagem focada na otimização e na velocidade da execução de algoritmos.

O contexto de utilizar pilha e fila se dá em sua implementação rápida, fácil e útil, podendo assim dizer que são as estruturas de dados mais fáceis de se compreender. Diversos algoritmos se faz o uso destas estruturas de dados [2] [4], nas quais os elementos são dispostos em função de regras que regulamenta a entrada e a saída dos elementos.

II. Teoria

Em 1955, Samelson e Bauer propuseram a ideia de pilha e em 1988 Bauer recebeu uma premiasão pela ivenção do princípio da pilha [7]. Pode-se dizer que filas é muito mais antigas do que pilhas, pois a teoria das filas se originou-se na pesquisa de Agner Krarup Erlang [8] quando descreveu um modelo para a central telefônica de Copenhague.

Pilha e fila são baseadas em uma regra cada uma. Na pilha tem-se o critério *Last In, First Out* (LIFO), sendo o último elemento a entrar deve ser o primeiro elemento a sair, e na fila tem-se o critério *First In, First Out* (FIFO), sendo o primeiro elemento a entrar deve ser o primeiro elemento a sair.

Uma pilha é uma sequência de elementos, no qual pode ser interpretada como: $S = [a_1, a_2, ..., a_n]$, em que a_n será o elemento mais recente da sequência dada. As operações básicas que essa sequência irá utilizar são: inserir um elemento no topo, retirar um elemento do topo, retornar o elemento que está no topo e verificar se a sequência está vazia.

Adotando a mesma sequência de elementos da pilha, pode-se interpretar uma fila sendo: $F = [a_1, a_2, ..., a_n]$, em que a_1 será o primeiro elemento da sequência e a_n será o último elemento, juntamente com as operações básicas, sendo elas: inserir um elemento ao final, retirar um elemento do início, retornar o elemento que está no início e verificar se a fila está vazia.

III. Proposta e Implementação

Este artigo propõe utilizar a linguagem *C++* para a implementação das estruturas pilha e fila. Toda essa implementação foi feita em um computador com sistema operacional *Linux* e com o compilador *GCC*, a versão da linguagem utilizada foi a *C++11*. As duas estruturas tem em sua implementação a programação orientada a objetos (POO), onde visa a construção de classes e métodos.

Implementou-se duas classes[1], sendo essas a classe *Stack* (implementação da estrutura pilha) e a classe *Queue* (implementação da estrutura fila). Na classe *Stack* foi efetuada as operações básicas mencionada na Seção II, essas operações são os métodos de classe, portanto, denominou-se os métodos *push*, *pop*, *top* e *empty*. Na classe *Queue* também tem-se os métodos com base nas operações da fila e esses métodos foram denominados *enqueue*, *dequeue*, *peek*, *empty*. Nas Figuras 1 e 2, pode-se visualizar o diagrama UML das classes *Stack* e *Queue*. Além das duas classes também foi implementado uma classe de exceção.

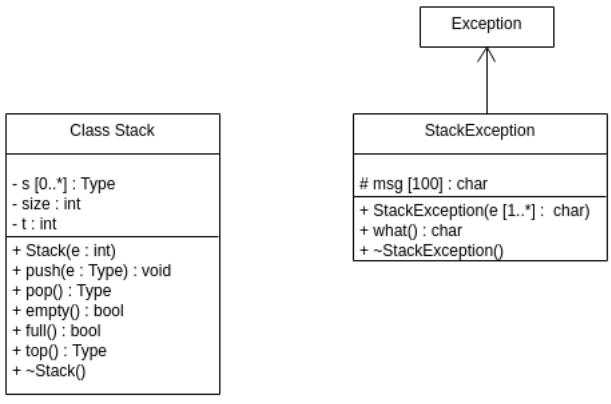

Figura 1. UML da classe *Stack*

[1]https://github.com/buzutilucas/scientific-programming/tree/master/Ex01

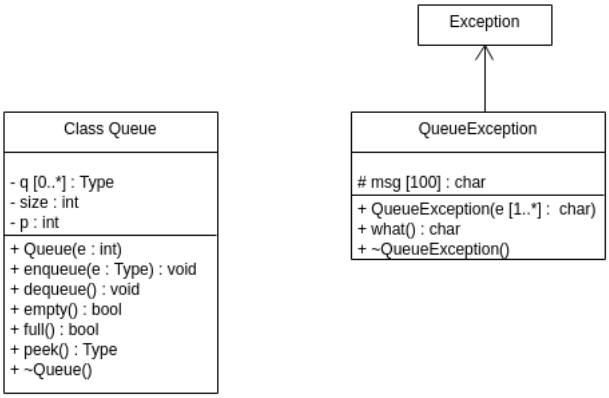

Figura 2. UML da classe *Queue*

IV. Experimentos e Resultado

Para testar a proposta desse artigo foi codificado um arquivo *main* para cada estrutura de dados, o arquivo foi escrito em *C++*. Nesse arquivo foi introduzido estruturas para testar os métodos de classe, porém, as entradas foram feitas manualmente, mas podendo ser feitas automaticamente e testada por um serviço de integração contínua hospedado, usado para criar e testar projetos de software. Os resultados do teste pode ser visualizados nas Figuras 3 e 4.

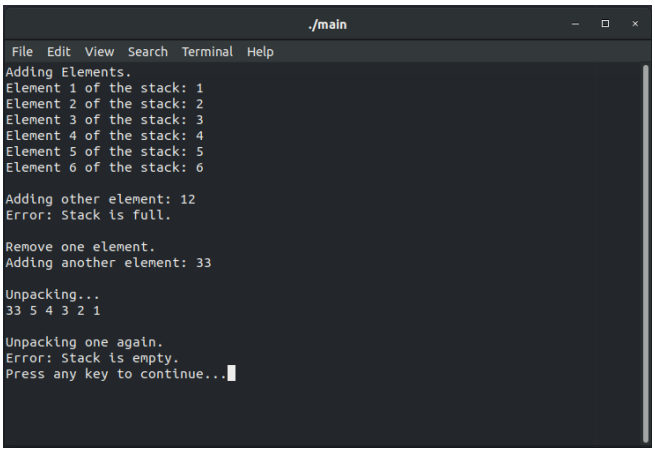

Figura 3. Resultados dos testes da classe *Stack*

Os métodos implementados nas classes abordada nesse artigo não sofreram erros de compilação, pois os métodos de classe foram projetados para sofrerem tratamento de erros. Em outras palavras, se o usuário da classe tentar adicionar ou remover um elemento na condição em que a pilha ou a fila esteja cheia ou vazia, uma mensagem é impressa, sendo assim evitando a interrupção do código por erro de compilação.

Analisando as duas estruturas conclui-se que cada uma tem a sua funcionalidade, sendo que não há uma comparação de qual é melhor ou pior.

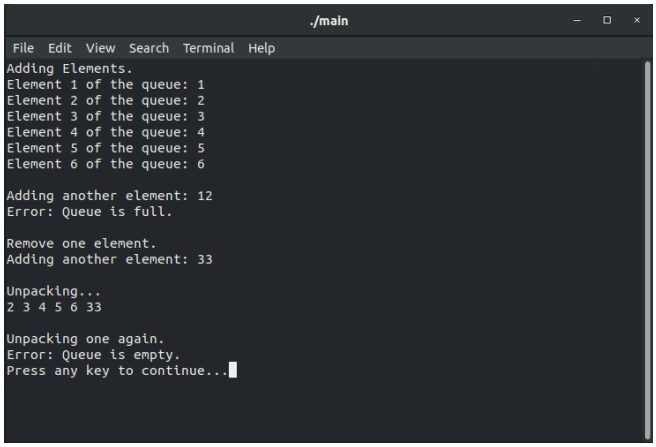

Figura 4. Resultados dos testes da classe *Queue*

V. Trabalhos Correlatos

Mesmo pilha e fila sendo estruturas de dados antigas, trabalhos relevantes nos dias atuais e até mesmo trabalhos de importância para os dias atuais propostos na década de 1990, 1980 ou 1970, usufruem nessas estruturas de dados [3] [5]. Algoritmos modernos de inteligência artificial e visão computacional utiliza-se das teorias ou das próprias estruturas de dados em sua modelação matemática [1] [6].

VI. Conclusão

Nesse artigo pode compreender e implementar as estruturas de dados pilha e fila. Além da compreensão uma análise crítica entre as duas estruturas foi proposta. Não há uma comparação entre melhor ou pior das estruturas, pois cada estrutura de dados terá o melhor enquadramento para resolver um determinado problemas. Vislumbra-se, como trabalhos futuros, a utilização dessas estruturas de dados em algoritmos modernos tal como *Deep Learning*.

Referências

[1] Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews and Noah A. Smith. “Transition-Based Dependency Parsing with Stack Long Short-Term Memory.” ACL (2015).
[2] R M Haralick and L G Shapiro, “Survey: Image segmentation techniques”, CVGIP, Vol. 29, pp 100-132, 1985.
[3] Joulin, Armand and Tomas Mikolov. “Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets.” NIPS (2015).
[4] Zhou, Rong and Eric A. Hansen. “Breadth-First Heuristic Search.” KR (2004).
[5] Rasley, Jeff, Konstantinos Karanasos, Srikanth Kandula, Rodrigo Fonseca, Milan Vojnovic and Sriram Rao. “Efficient queue management for cluster scheduling.” EuroSys (2016).
[6] Yang, Hao, Hesham A. Rakha and Mani Venkat Ala. “Eco-Cooperative Adaptive Cruise Control at Signalized Intersections Considering Queue Effects.” IEEE Transactions on Intelligent Transportation Systems 18 (2016): 1575-1585.
[7] Bauer, Friedrich L., et al. ”Formal program construction by transformations-computer-aided, intuition-guided programming.”IEEE Transactions on Software Engineering 15.2 (1989): 165-180.
[8] Sundarapandian, Vaidyanathan. Probability, statistics and queuing theory. PHI Learning Pvt. Ltd., 2009.