CENTRO UNIVERSITÁRIO FEI

VICTOR BIAZON

RA: 119.115-4

**RELATÓRIO II – HERANÇA CLASSES.**

SÃO BERNARDO DO CAMPO

2019

VICTOR BIAZON

RA: 119.115-4

**RELATÓRIO II – HERANÇA CLASSES.**

Relatório de desenvolvimento de classes de listas, filas e pilhas orientadas a objeto utilizando herança de classes, desenvolvido pelo aluno Victor Biazon, RA 119.115-4, para disciplina PEL216 – Programação Cientifica, ministrada pelo professor Reinaldo Bianchi.

São Bernardo do Campo

2019

Sumário

[Motivação: 4](#_Toc11506177)

[Objetivo: 4](#_Toc11506178)

[Teoria: 5](#_Toc11506179)

[Implementação: 7](#_Toc11506180)

[Experimentos e resultados: 10](#_Toc11506181)

[Códigos: 12](#_Toc11506182)

[Referências bibliográficas: 17](#_Toc11506183)

Motivação: Utilização e pratica do uso de orientação a objetos para a disciplina de programação cientifica.

# Objetivo:

Desenvolver as estruturas de dados pilhas e filas com programação orientada a objeto.

Utilizando a linguagem C++ de forma que seja possível a criação de classes e objetos necessários para a representação das estruturas de dados mencionadas acima e baseado no conceito de pilha finita e fila wrapped-around.

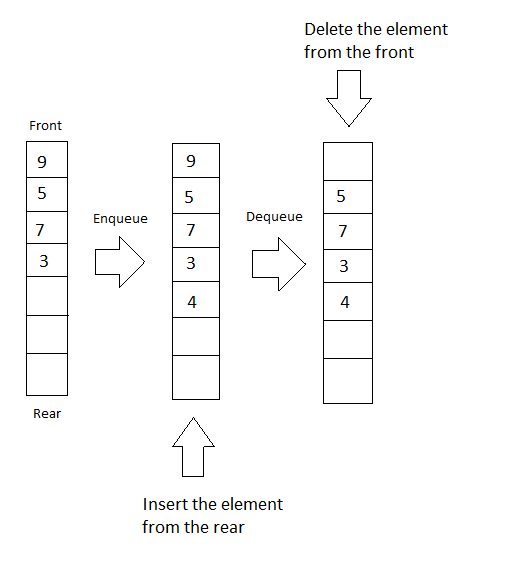
# Teoria:

**Pilhas** – são estruturas de dados baseadas no sistema de armazenagem de dados LIFO (Last In First Out), ou seja os últimos dados a entrar são os primeiros a sair. Filas (stacks) geralmente tem um limite de armazenagem e portanto quando o limite é excedido ocorre o “stack overflow”. A pilha precisa de um array para armazenagem dos elementos a serem inseridos e uma variável auxiliar para armazenar a última posição com um elemento inserido. As operações possíveis são geralmente Push() para inserir um valor no array e Pop() para retirar um valor do array. Toda vez que se insere um valor na pilha se incrementa index que guarda a posição do fim da pilha, e toda vez que se retira um elemento o index é decrementado. A pilha também tem a função Top() que mostra o valor no topo da fila sem retirá-lo.

[1]

**Filas** – são estruturas de dados baseadas no sistema de armazenagem de dados FIFO (First In First Out), ou seja os primeiros dados a entrar são os primeiro a sair. As filas (queues) tem limite de armazenagem que ocorre quando todos os elementos da fila estão ocupados. Ao se retirar e colocar novos elementos a posição inicial e final se desloca, como o fim da fila tem um limite usa-se a configuração wrapped-around para voltar ao início e recomeçar a armazenagem.

As funções possíveis para as filas são Enqueue() para inserir um elemento na fila, Dequeue() para retirar um valor da fila e Frente() para verificar qual o primeiro valor na frente da fila. Também precisa de duas variáveis auxiliares uma para guardar o início da fila, e uma para guardar o fim da fila. Toda vez que se insere um valor na fila se incrementa o valor que guarda o index do início da fila e toda vez que se retira um valor da fila se incrementa o index que guarda a posição do fim da fila.

[1]

# Implementação:

Para construção da fila e da pilha foram criadas duas classes respectivamente chamadas de Pilha e Fila.

**A classe Pilha** terá como variáveis privadas um array que armazenará os elementos e uma variável index que armazenará a posição do topo da pilha. E como funções públicas terá as funções Push() e Pop(), para inserir e retirar elementos, respectivamente. Resultando na seguinte UML:

|  |
| --- |
| Pilha |
| Privado:  Int Vetor[Limite]  Int Index |
| Publico:  Void Push(int x)  Int Pop()  Int Top() |

O pseudo-código da função Push(x) é:

Se Index < Limite:

Vetor[Index] <= x;

Index = Index + 1;

Se (Index + 1 = Limite):

Imprime “Pilha cheia”;

O pseudo-código da função Int Pop() é:

Se Index > 0:

Retira o valor do Vetor[Index];

Index = Index - 1;

Se (Index < 0):

Imprime “Pilha Vazia”;

O pseudo-código da função Int Top() é:

Se Index > 0:

Mostra o valor do Vetor[Index];

Se não:

Imprime “Pilha vazia”;

**A classe Fila** terá como variáveis privadas um array que armazenará os elementos, uma variável indexIn que armazenará a posição do fim da fila, uma variável indexou que armazenará a posição do início da fila e uma variável PosOcup que armazenará o número de posições ocupadas no array. E como funções públicas terá as funções Enqueue() e Dequeue(), para inserir e retirar elementos, respectivamente, também terá a função ValorFrente() que mostra o valor no início da fila sem retirar o valor desta.

|  |
| --- |
| Fila |
| Privado:  Int Vetor[Limite]  Int IndexIn  Int IndexOut  Int PosOcup |
| Público:  Void Enqueue(int x)  Int Dequeue()  Int ValorFrente() |

Pseudo-código da função Void Enqueue(x):

Se (PosOcup = Limite):

Imprime “Fila cheia”;

Ou então se (Index + 1 = Limite e PosOcup < Limite):

IndexIn = 0;

Insere valor x no Vetor[indexIn];

Incrementa indexIn;

Incrementa PosOcup;

Ou então:

Insere valor x no Vetor[indexIn];

Incrementa indexIn;

Incrementa PosOcup;

Pseudo-código da função Int Dequeue():

Se (PosOcup = 0):

Imprime “Fila vazia”;

Ou então se (Index + 1 = Limite):

Retira valor x do Vetor[indexOut];

IndexOut = 0;

Incrementa indexIn;

Decrementa PosOcup;

Ou então:

Retira valor x do Vetor[indexOut];

Incrementa indexIn;

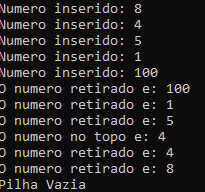
Decrementa PosOcup;

Pseudo-código da função Int ValorFrente(x):

Retorna o valor de Vetor[IndexIn - 1];

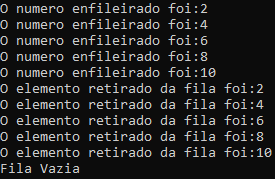
# Experimentos e resultados:

Para testar o código da pilha foi executado uma sequência de cinco inserções Push() e seis retiradas Pop() seguidas. Para seguir a execução foi utilizada a sequência de cinco inserções e em seguida seis retiradas tendo por último a frase “Pilha vazia”.



A execução mostrou que os valores que foram inseridos por último saíram primeiro ou seja, na ordem contraria à inserção.

Para testar o código da fila foi criado uma verificação de função onde para a ação com o número 1 se executa o Enqueue e solicita ao usuário que o mesmo digite um número inteiro, para a função com o número 2 se executa o Dequeue e mostra o valor retirado da pilha.



A execução mostrou que para 4 inserções consecutivas dos números de 1 a 4 foram retirados na mesma ordem de 1 a 4 e quando houve a tentativa de retirar um novo valor com a pilha vazia o programa retornou “Pilha vazia.”

# Códigos:

Fila.h:

class Fila

{

#define Lim 100 // define limite do array

public:

void Enqueue(int x); //função enfileirar elemento x

int Dequeue(); // função desenfileirar

bool FilaVazia(); //função verifica fila vazia

int ValorFrente(); //função mostra o valor na frente da fila

private:

int Vetor[Lim] = { 0 }; //definição e inicialização do array dos elementos

int indexIn = 0; // Index da frente da fila

int indexOut = 0; //Index do fim da fila

int PosOcup = 0; // armazena quantidade de posições ocupadas na fila

};

Fila.cpp:

#include "Fila.h"

#include <iostream>

using namespace std;

// Configuração de fila wrapped around

bool Fila::FilaVazia() {

if (PosOcup == 0) { //se a quantidade de posições ocupadas for 0 retorna True

return true;

}

else { // se nao retorna False

return false;

}

}

int Fila::ValorFrente() {

if (not(FilaVazia())) { //se a fila não estiver vazia, retorna o valor do elemento na frente da fila

return Vetor[indexIn - 1];

}

else { // se estiver retorna o texto Nao ha elementos na fila

cout << "Não há elementos na fila" << endl;

}

}

void Fila::Enqueue(int x) {

if (PosOcup == Lim) { //se a quantidade de posições ocupadas for igual ao limite, retorna Fila cheia

cout << "Fila cheia" << endl;

}

else if(indexIn + 1 == Lim){ // se o index de entrada chegar no final da fila ele volta ao início e começa de novo

indexIn = 0; // reinicializa o index da frente da fila

Vetor[indexIn] = x; //guarda o elemento

PosOcup++; //incrementa número de posições ocupadas

}

else { // se nao

Vetor[indexIn] = x; //insere elemento na fila

indexIn++; //incrementa index do início da fila

PosOcup++; //incrementa número de posições ocupadas

}

}

int Fila::Dequeue() {

int mem = 0; //inicializa memoria auxiliar

if(PosOcup <= 0){ //se o número de posições ocupadas for menor ou igual a 0

indexOut = 0; //reinicializa index de fim da fila

indexIn = 0; //reinicializa index de início da fila

cout << "Fila Vazia" << endl; // imprime texto fila vazia

return 0;

}

else if (indexOut + 1 == Lim and PosOcup > 0) { //se o próximo elemento é o limite da fila e há elementos na fila

indexOut = 0; //reinicializa index do fim da fila

mem = Vetor[indexOut]; //guarda elemento na memória auxiliar

Vetor[indexOut] = 0; //zera posição atual do fim da fila

PosOcup--; //decrementa número de posições ocupadas

return mem; //retorna valor do elemento retirado

}

else if (indexOut + 1 >= 0 and PosOcup > 0) { //se o próximo elemento não é o limite da fila e há elementos na fila

mem = Vetor[indexOut]; //guarda elemento na memoria auxiliar

Vetor[indexOut] = 0; //zera posição atual do fim da fila

PosOcup--; //decrementa número de posições ocupadas

indexOut++; //incrementa index do fim da fila

return mem; //retorna valor do elemento retirado

}

}Pilha.h:

class Pilha

{

#define Lim 100 // limite do tamanho do array

public:

void Push(int x); // função inserir elemento na pilha

int Pop(); // função retirar elemento da pilha

int Top(); // função mostrar elemento no topo da pilha

private:

int Vetor[Lim] = {0}; // array de armazenamento dos elementos

int index = -1; // index de contagem da posição do topo da pilha

};

Pilha.cpp:

#include <iostream>

#include "Pilha.h"

using namespace std;

void Pilha::Push(int x) {

if (index + 1 == Lim) { // verifica se o valor do topo da pilha + 1 elemento é igual ao limite do array

cout << "Pilha cheia" << endl; //se sim, imprime indicando pilha cheia

}

else{ //se a pilha não estiver cheia

index++; //incrementa a posição do topo

Vetor[index] = x; //insere elemento

}

}

int Pilha::Pop() {

if (index >= 0) { //verifica se a pilha não esta vazia

int mem; //define memoria auxiliar

mem = Vetor[index]; //armazena o valor do elemento na memória auxiliar

Vetor[index] = 0; //zera o elemento

index--; //decrementa a posição do topo

return mem; //retorna valor armazenado

}

else {

cout << "Pilha Vazia" << endl; // se estiver vazia imprime Pilha vazia

return 0;

}

}

int Pilha::Top() {

if (index >= 0) { //se houverem elementos na pilha

return Vetor[index]; //retorna o valor do primeiro elemento no topo

}

else {

cout << "Pilha vazia" << endl; // se não retorna pilha vazia

}

# }Trabalhos correlatos:

Implementing Lock-Free Queues John D. Valois Department Of Computer Science Rensselaer Polytechnic Institute Troy, Ny 12180

[Http://People.Cs.Pitt.Edu/~Jacklange/Teaching/Cs2510-F12/Papers/Implementing\_Lock\_Free.Pdf](http://people.cs.pitt.edu/~jacklange/teaching/cs2510-f12/papers/implementing_lock_free.pdf)

# The Broker Queue: A Fast, Linearizable FIFO Queue For Fine-Granular Work Distribution On The GPU

# [Https://Markussteinberger.Net/Papers/Brokerqueue.Pdf](https://markussteinberger.net/papers/BrokerQueue.pdf)

Teaching Software Testing using Data Structures

<https://pdfs.semanticscholar.org/e166/17bbb684af62ea9aed6863f3b5827de7bdd2.pdf>

A Scalable, Correct Time-Stamped Stack

<http://delivery.acm.org/10.1145/2680000/2676963/p233-dodds.pdf?ip=189.100.27.140&id=2676963&acc=OA&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E36B65A2A0B97900A&__acm__=1560625848_e8bfbdf4877193982758cf4d27a0c081>

# Conclusão:

Os resultados para a implementação de programação orientada a objetos das estruturas de dados Filas e Pilhas ocorreram de acordo com o esperado e apresentado na teoria, onde a Fila se comportou como FIFO e a Pilha se comportou como LIFO.

A orientação a objetos se mostrou muito útil onde a criação de classes e objetos facilita o uso de funções e atributos de cada um dos objetos de forma a ser possível utilizar vários objetos com os mesmos atributos dinamizando o uso dos mesmos sem ser necessário declará-los inicialmente no programa o uso de cada um, criando e destruindo cada objeto de acordo com a necessidade do programa.

Por limitações de memória inerentes aos limites de hardware tornam o uso de ambos restringidos por estes. Desta forma não é possível implementar filas ou pilhas infinitas devido a limites físicos de espaço.

# Referências bibliográficas:

[1] <https://www.hackerearth.com/pt-br/practice/notes/stacks-and-queues/>