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**

Objetivo: **4**

Teoria: **5**

Implementação: **6**

Experimentos e resultados: **11**

Códigos: **13**

Trabalhos correlatos: **18**

Conclusão: **19**

Referências bibliográficas: **20**

MOTIVAÇÃO: Utilização e pratica do uso de orientação a objetos para a disciplina de programação cientifica utilizando os conceitos de herança, polimorfismo e abstração.

OBJETIVO:

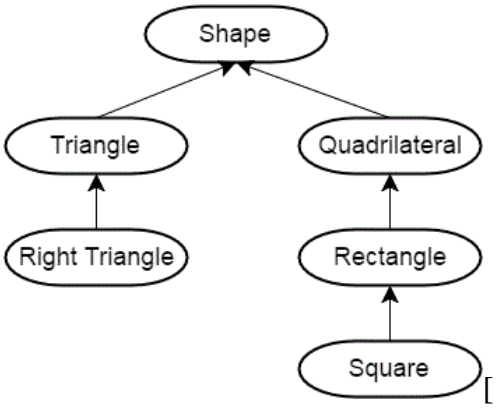
Desenvolver as estruturas de dados listas ligadas, pilhas e filas com programação orientada a objeto, sendo as pilhas e filas utilizando herança de classe da classe lista.

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 herança, ou seja reaproveitando métodos da classe lista para implementar as classes fila e pilha.

TEORIA:

A Herança se baseia no conceito de que classes filhas herdam métodos e parâmetros de suas classes pai, desta forma pode-se criar classes dependentes reaproveitando o código de forma organizada e estruturada.

Como mostrado na figura abaixo, se fossemos criar classes de objetos como formas geométricas, poderíamos criar, por exemplo, as subclasses que tem três e quatro lados respectivamente. Em seguida poderíamos criar subclasses destas, especificando ainda mais o objeto, onde na hierarquia das classes ficaria sendo: um quadrado é um retângulo, e é um quadrilátero e é uma forma. Sendo o quadrado herdando todos os métodos das classes que ele é dependente.



[1]

IMPLEMENTAÇÃO:

Para construção da lista, fila e da pilha foram criadas uma classe Lista e duas subclasses respectivamente chamadas de Pilha e Fila.

A classe Lista implementará a maior parte dos métodos que as classes Pilha e Fila utilizarão, e terá a seguinte UML:

Lista

Lista();
~Lista();
void InsereFinal(int x);
void InsereInicio(int
int RetiraFinal();
int RetiraInicio();
Elemento* Busca(int x);
void Imprimir();
Elemento* GetIni();
Elemento* GetFim();
bool isEmpty();

Private: Elemento* ini; Elemento* fim

O pseudo-código do construtor Lista() é;

Ini = NULL;

Fim = NULL;

//Inicializa ambos os ponteiros.

O pseudo-código do destrutor ~Lista() é;

Cria-se um elemento temporário:

```
Elemento* tmp;
Tmp <- ini; //Tmp recebe o ponteiro do início da lista

while (tmp != NULL) { //Enquanto tmp for diferente de NULL)
        ini <- proximo; //Atualiza o elemento da frente da fila
        delete tmp; //Apaga o elemento que esta endereçado como tmp
        tmp <- ini; //Atualiza posição de tmp para a próxima posição.
        }
```

O pseudo-código da função InsereFinal() é:

```
    Cria Elemento = novo;
    Novo->Valor = x;
Se fim == NULL:
    InsereInicio(x);
Se não: fim.proximo = novo;
    Novo.anterior = fim;
    Fim = novo;
```

O pseudo-código da função InsereInicio() é:

```
    Cria Elemento = novo;
    Novo->Valor = x;
Se ini == NULL:
    Ini = fim = novo;
```

O pseudo-código da função RetiraFinal() é:

```
    Auxiliar = fim.valor;
Se (fim.anterior == NULL):
    Fim <= NULL
    Ini <= NULL;
Se não:
    Fim.anterior.proximo <= NULL;
    Fim = Fim.Anterior;
    Auxiliar = fim.valor;
```

O pseudo-código da função RetiraInicio() é:

```
    Auxiliar = ini.valor;
Se (ini.proximo == NULL):
        Fim <= NULL
        Ini <= NULL;
Se não:
    Ini.proximo.anterior <= NULL;
    Ini = Ini.proximo;
    Auxiliar = ini.valor;
```

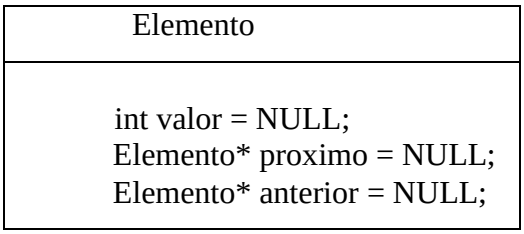
O pseudo-código da função Busca(x) é:

```
    Cria Elemento Tmp; //
    Tmp <= Ini; //O ponteiro temporário recebe a posição do inicio.
Enquanto (Tmp != NULL):
    Se (Tmp.Valor == x):
        Retorna Tmp;
    Tmp = Tmp.proximo;
Se não encontrar:
    Retorna NULL;
```

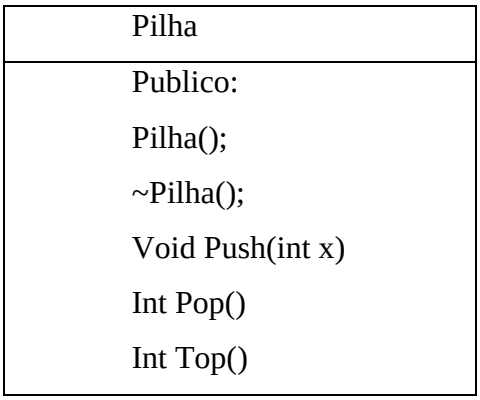
O pseudo-código da função Imprimir(x) é:

```
Cria Elemento Tmp; //
    Tmp <= Ini; //O ponteiro temporário recebe a posição do inicio.
Enquanto (Tmp != NULL):
        Imprime Tmp.Valor;
        Tmp <=Tmp.proximo;
```

O elemento será uma struct com a seguinte UML:



A classe Pilha será herdeira da classe Lista e resultara na seguinte UML:



O pseudo-código do construtor Pilha() é;

Call Lista(); //Chama o construtor da Lista.

O pseudo-código do construtor Pilha() é;

Call ~Lista(); //Chama o destrutor da Lista.

O pseudo-código da função Push(x) é:

InsereFinal(x); //Chama método da lista para inserir no fim da lista

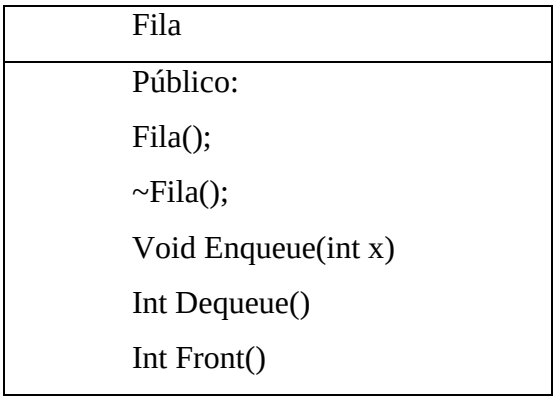
O pseudo-código da função Int Pop() é:

Retorna RetiraFinal(); // Retorna o valor da retirada do final da lista.

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

Elemento * Aux = GetFim(); //armazena o ponteiro do último elemento da lista.
Retorna Aux->Valor; //Mostra o valor do último elemento da lista

A classe Fila será herdeira da classe Lista e resultara na seguinte UML:



O pseudo-código do construtor Fila() é;

Call Lista(); //Chama o construtor da Lista.

O pseudo-código do construtor Fila() é;

Call ~Lista(); //Chama o destrutor da Lista.

Pseudo-código da função Void Enqueue(x):

InsereInicio(x); //Insere valor no início da fila

Pseudo-código da função Int Dequeue():

Retorna RetiraFinal(); // Retorna o valor da função RetiraFinal da lista

Pseudo-código da função Int Front(x):

Elemento * Aux = GetFim(); //Armazena o ponteiro do último elemento da lista.
Retorna Aux->Valor; //Mostra o valor do último elemento da lista

EXPERIMENTOS E RESULTADOS:

Para testar as classes e subclasses criadas foram feitos os seguintes testes:

Lista: Foram inseridos seis números seguidos, em seguida retirado o número do início e o número do final e impresso todos seus elementos retornando o seguinte resultado:

```
Teste da Lista:
Inserido valor: 1
Inserido valor: 2
Inserido valor: 3
Inserido valor: 4
Inserido valor: 5
Inserido valor: 6
Retirado do inicio: 1
Retirado do final: 6
A lista armazenada e: 2 - 3 - 4 - 5 -
```

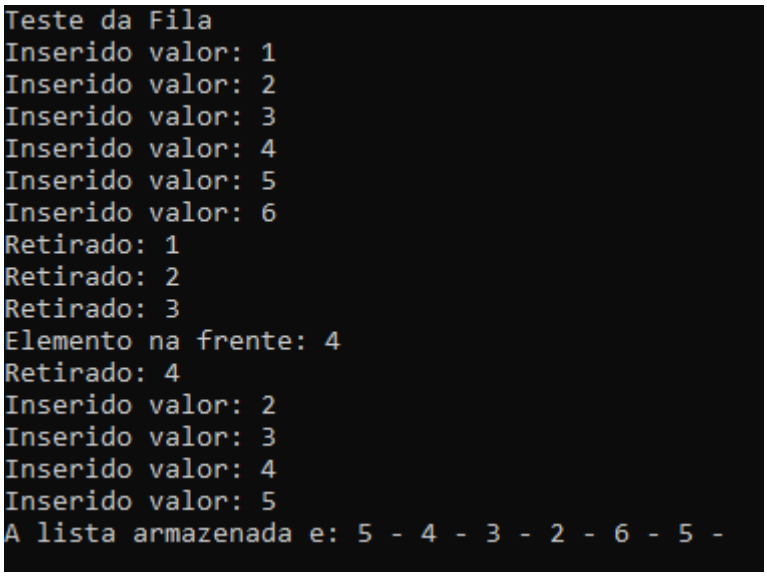
Como pode-se observar os números retirados 1 e 6, não constaram mais na impressão após terem sido retirados.

Pilha: Foram inseridos quatro números seguidos e em seguida retirados da pilha, depois inseridos quatro números novamente, retirado um e inserido mais um, por último foi impressa a pilha restante.

```
Teste da Pilha:
Inserido valor: 1
Inserido valor: 2
Inserido valor: 3
Inserido valor: 4
Retirado: 4
Retirado: 3
Retirado: 2
Elemento no topo: 1
Retirado: 1
Inserido valor: 1
Inserido valor: 2
Inserido valor: 3
Inserido valor: 4
Retirado: 4
Inserido valor: 5
A lista armazenada e: 1 - 2 - 3 - 5 -
```

Como pode-se notar os primeiros números inseridos foram todos retirados, em seguida o número 4 que foi retirado da série seguinte de inserções não constou na impressão final, pois foi retirado do todo da pilha.

Fila: Foram inseridos seis números seguidos e em seguida retirados quatro, inseridos mais quatro números e impressa fila toda.



Como pode-se notar os números “5” e “6” que estão na frente da fila fazem parte da primeira inserção. E em seguida do 2 ao 5 foram inserido na fila.

CÓDIGOS:

Elemento.h

```
#define NULL 0

struct Elemento // cria struct de elemento com ponteiros de anterior e proximo
{
	int valor = NULL;
	Elemento* proximo = NULL;
	Elemento* anterior = NULL;

	 };
```

Lista.h

```
class Lista
{
public:
	Lista(); //Construtor
	~Lista(); //Destrutor
	void InsereFinal(int x); //Insere elemento no final da lista
	void InsereInicio(int x);// Insere elemento no início da lista
	int RetiraFinal(); //Retira elemento do final da lista
	int RetiraInicio(); //retira elemento do início da lista
	Elemento* Busca(int x); //Retorna ponteiro do elemento que contem x, se não
retonra NULL
	void Imprimir(); //Imprime a lista toda
	Elemento* GetIni(); //Retorna ponteiro da posição inicial da lista
	Elemento* GetFim(); //Retorna ponteiro da posição final da lista
	bool isEmpty(); //retorna true para lista vazia, false para lista não fazia.

private:
	Elemento* ini; //Guarda o ponteiro do início da lista ligada
	Elemento* fim; //Guarda o ponteiro do fim da lista ligada

};
```

Lista.cpp

```
#include "Lista.h"
#include <stdio.h>
#include <iostream>
#include <stdexcept>
#define NULL 0
using namespace std;

Lista::Lista() // construtor da lista
{
	ini = NULL;
	fim = NULL;
}

Lista::~Lista() //destrutor da lista
{
	Elemento* tmp;

	tmp = ini;
	while (tmp != NULL) {
		ini = ini->proximo; //Atualiza o elemento da frente da fila
```

```
			delete tmp; // deleta elemento temporário
			tmp = ini; // atualiza tmp para próxima posição
		}
}

void Lista::InsereFinal(int x) { //insere elemento no fim da lista

		if (fim == NULL) { //se não houver ponteiro para o elemento do fim da lista,
esta vazio, chama função insereInicio.
			InsereInicio(x);
		}
		else {
			Elemento* novo;
			novo = new Elemento; // cria novo elemento no ponteiro novo
			novo->valor = x; //atribui o valor x ao valor do novo elemento
			cout << "Inserido valor: " << x << endl;
			fim->proximo = novo; //atualiza a posição proximo do último elemento para
o novo elemento
			novo->anterior = fim; // atualiza a posição do elemento anterior do novo
elemento para o elemento que era o ultimo
			fim = novo; //atribui a posição de fim para o novo elemento
		}
}

void Lista::InsereInicio(int x) { //insere elemento no início da lista
		Elemento* novo;
		novo = new Elemento; // cria novo elemento no ponteiro novo
		novo->valor = x; //atribui o valor x ao valor do novo elemento
		cout << "Inserido valor: " << x << endl;
		if (ini == NULL) { //Se não houver ponteiro para o primeiro elemento da lista,
significa que esta vazia logo o novo elemento é o primeiro e o ultimo
			ini = fim = novo;
		}
		else {
			novo->proximo = ini; //atualiza a posição do proximo elemento do novo
elemento como o elemento inicial
			ini->anterior = novo; // atualiza a posição do anterior do elemento
inicial para o novo elemento
			ini = novo; // o novo elemento é agora o elemento inicial
		}
}

int Lista::RetiraFinal() { //retira valor do final da lista
		if (isEmpty()) { //Verifica se a lista está vazia, se estiver lança exceção
			throw std::exception("Lista vazia");
		}
		int aux = fim->valor; //guarda valor do elemento final numa variável auxiliar
		if (fim->anterior == NULL) { //se o elemento anterior ao fim é nulo, só há um
elemento na lista, logo se inicializa os ponteiros fim e ini
			fim = NULL;
			ini = NULL;
			return aux;
		}
		else {
			fim->anterior->proximo = NULL; //se não atualiza o valor do ponteiro
proximo do elemento anterior ao final para null
				fim = fim->anterior; // atualiza o elemento anterior sendo o novo
final da lista
		}

		return aux; //retorna valor armazenado
}
```

```
int Lista::RetiraInicio() { //retira valor do início da lista

	if (isEmpty()) { //Verifica se a lista está vazia, se estiver lança exceção
		throw std::exception("Lista vazia");
	}
	int aux = ini->valor; //guarda valor do elemento final numa variável auxiliar
	if (ini->proximo == NULL) {//se o elemento próximo ao início é nulo, só há um
elemento na lista, logo se inicializa os ponteiros fim e ini
		fim = NULL;
		ini = NULL;
		return aux;
	}
	ini->próximo->anterior = NULL; //se não atualiza o valor do ponteiro anterior do
elemento próximo ao início para null
	ini = ini->proximo; //atualiza o próximo elemento sendo o início da lista
	return aux; //retorna valor armazenado

}

Elemento* Lista::Busca(int x) { //verifica se existe um valor armazenado na lista e
retorna o ponteiro para a posição caso exista
	if (isEmpty()) { //Verifica se a lista está vazia, se estiver lança exceção
		throw std::exception("Lista vazia");
	}
	Elemento* tmp = ini; //atribui um elemento temporário para o início da lista
	do {
		if (tmp->valor == x) { // verifica se o elemento coincide com o valor
buscado
			return tmp; // se sim retorna ponteiro da posição encontrada
		}
		tmp = tmp->proximo; //atualiza posição do ponteiro para próximo elemento
	} while (tmp != NULL); // executa enquanto não chegar no final
	return NULL; //se não for encontrado retorna NULL

}

void Lista::Imprimir() { //imprime do começo ao fim os valores da lista
	if (isEmpty()) { //Verifica se a lista está vazia, se estiver lança exceção
		throw std::exception("Lista vazia");
	}
	Elemento* tmp = ini; //atribui um elemento temporário para o início da lista
	cout << "A lista armazenada e: ";
	do {
		cout << tmp->valor << " - ";
		tmp = tmp->proximo; //atualiza posição do ponteiro para proximo elemento
	} while (tmp != NULL); // executa enquanto não chegar no final
	cout << endl;

}

Elemento* Lista::GetIni() // Retorna ponteiro do início da lista
{
	return ini;
}

Elemento* Lista::GetFim() // Retorna ponteiro do final da lista
{
	return fim;
}
```

```
bool Lista::isEmpty() // verifica lista vazia
{
	return ini == NULL; //se o ponteiro do início estiver vazio, a lista está vazia, retorna true

}
```

Pilha.h

```
#include "Lista.h"
class Pilha : public Lista //Pilha herda atributos de Lista
{
public:
	Pilha(); //Construtor
	~Pilha(); // Destrutor
	void Push(int x); // Empilha valor na pilha criada
	int Pop(void); // Desempilha valor da pilha criada
	int Top(void); // Mostra o valor no topo da pilha

};
```

Pilha.cpp

```
#include "Pilha.h"
#include "Lista.h"
#include <iostream>
#include <stdexcept>
using namespace std;

Pilha::Pilha()
{
	Lista::Lista(); // Chama construtor da lista
}

Pilha::~Pilha()
{
	Lista::~Lista(); //chama destrutor da lista
}

void Pilha::Push(int x) // insere valor no final da pilha pelo método herdado
{
	InsereFinal(x);
}

int Pilha::Pop(void) //retira valor no final da pilha pelo método herdado
{
		int aux = RetiraFinal();
		return aux;
	}

int Pilha::Top(void) //mostra valor no topo da pilha
{
	Elemento* aux = GetFim();
	return aux->valor;
}
```

Fila.h

```
#include "Lista.h"
class Fila : public Lista //Fila herda atributos de Lista
{
public:
	Fila(); // Construtor
	~Fila(); // Destrutor
	void Enqueue(int x); // Insere valor no topo da pilha
	int Dequeue(); // Retira valor do topo da pilha
	int Front(); // Mostra valor no topo da pilha sem retirar
};
```

Fila.cpp

```
#include "Fila.h"

Fila::Fila()
{
	Lista::Lista();// Chama construtor da lista
}

Fila::~Fila()
{
	Lista::~Lista(); //chama destrutor da lista
}

void Fila::Enqueue(int x)
{
	InsereInicio(x); //insere valor no início da fila pelo método herdado
}

int Fila::Dequeue()
{
	int aux = RetiraFinal(); //retira valor do final da fila pelo método herdado
	return aux;
}

int Fila::Front() //mostra valor na frente da fila
{
	Elemento* aux = GetFim();
	return aux->valor;
}
```

TRABALHOS CORRELATOS:

On the Interaction of Object-Oriented Design Patterns and Programming Languages Technical Report CSD-TR-96-020 Gerald Baumgartner" Konstantin Laufer..... Vincent F. Russo'" 1996

https://pdfs.semanticscholar.org/86fa/4a3ee3bab35581efa3e99c86797e0696e278.pdf

Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications

Felix Schuster et al; 2015

https://ieeexplore.ieee.org/abstract/document/7163058

CONCLUSÃO:

Os resultados para a implementação de programação orientada a objetos das estruturas de dados Filas e Pilhas através da herança da classe Lista ocorreram de acordo com o esperado e apresentado na teoria, onde a Fila se comportou como FIFO e a Pilha se comportou como LIFO, herdando os métodos da Lista e facilitando a implementação das mesmas por reaproveitar o código composto na Lista.

A orientação a objetos com herança se mostrou muito útil onde a criação de subclasses de objetos facilita o uso de funções e atributos já criados 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 novamente nas subclasses.

Para tratamento de exceções também foi implementado em C++ as funções Try e Catch para que fosse possível após ocorrer exceções na execução do programa que o usuário recebesse um diagnóstico mais preciso e claro e também a execução de uma rotina para tratar o erro.

REFERÊNCIAS BIBLIOGRÁFICAS:

[1] https://www.learncpp.com/cpp-tutorial/112-basic-inheritance-in-c/