Univiçosa – Centro Universitário de Viçosa
Cursos: Análise e Desenvolvimento de Sistemas e Engenharia de Computação
Disciplina: ADS101/ECO131 – Programação de Computadores 2

Aula 09 – Introdução a fila

1) Introdução

Uma fila é uma estrutura de dados onde os elementos são inseridos em uma extremidade (chamada de **final da fila**) e removidos da outra extremidade (chamada de **início da fila**). Por isso, ela segue a política **FIFO (First In, First Out)**, ou seja, o primeiro elemento a entrar é o primeiro a sair.

Principais Operações

- **Enqueue (inserir)**: Adiciona um elemento ao final da fila.
- **Dequeue (remover)**: Remove o elemento que está no início da fila.
- **Front (início)**: Acessa o primeiro elemento da fila (sem remover).
- **Empty (vazia)**: Verifica se a fila está vazia.

Observe um exemplo de fila em C++.

```
#include <iostream>
#include <queue> // incluímos a biblioteca para criar e manipular a fila

int main() {
    std::queue<int> fila;

    // Inserindo elementos na fila (Enqueue)
    fila.push(10);
    fila.push(20);
    fila.push(30);

    // Acessando o elemento da frente da fila
    std::cout << "Elemento no início da fila: " << fila.front() << std::endl;

    // Removendo o elemento da frente (Dequeue)
    fila.pop();

    std::cout << "Elemento no início após remoção: " << fila.front() << std::endl;

    // Verificando se a fila está vazia
    if (fila.empty()) {
```

```
        std::cout << "A fila está vazia!" << std::endl;
    } else {
        std::cout << "A fila não está vazia!" << std::endl;
    }

    return 0;
}
```

Esse código usa a biblioteca padrão `<queue>` para criar e manipular uma fila de inteiros. Você pode experimentar com outros tipos de dados e expandir a lógica conforme necessário.

2) Aplicando o uso da Fila em um sistema real

Um exemplo prático comum do uso de **Fila** é em um **sistema de impressão**.

Imagine um escritório onde várias pessoas enviam documentos para serem impressos em uma única impressora. A impressora precisa processar os pedidos na ordem em que chegam, ou seja, o primeiro documento enviado será o primeiro a ser impresso, e os seguintes entrarão na fila de espera. Isso é um perfeito caso de uso para a estrutura de dados Fila, pois respeita o comportamento **FIFO (First In, First Out)**.

```
#include <iostream>
#include <queue>
#include <string>

struct Documento {
    std::string nome;
    int paginas;

    Documento(std::string n, int p) : nome(n), paginas(p) {}
};

int main() {
    std::queue<Documento> filaImpressao;

    // Enviando documentos para a fila de impressão
    filaImpressao.push(Documento("Relatório Financeiro", 10));
    filaImpressao.push(Documento("Contrato de Trabalho", 5));
    filaImpressao.push(Documento("Apresentação de Vendas", 15));

    // Simulando o processo de impressão
    while (!filaImpressao.empty()) {
        Documento docAtual = filaImpressao.front();  // Pega o primeiro da fila
```

```
        std::cout << "Imprimindo: " << docAtual.nome
                  << " (" << docAtual.paginas << " páginas)" << std::endl;

        filaImpressao.pop();  // Remove o documento da fila após imprimir
    }

    return 0;
}
```

Explicação:

1. **Estrutura `Documento`**: Criamos uma struct `Documento` que possui o nome do documento e o número de páginas.
2. **Fila de Impressão**: A fila (`std::queue<Documento>`) armazena os documentos a serem impressos.
3. **Processo de Impressão**: Usamos um loop `while` para simular o processo de impressão. O documento da frente é acessado com `filaImpressao.front()` e removido após ser "impresso" com `filaImpressao.pop()`.

Neste exemplo, a fila garante que os documentos sejam impressos na ordem em que foram enviados, sem qualquer inversão.

Esse modelo é utilizado em muitos sistemas reais, como filas de processamento de tarefas ou mesmo em servidores web, que processam requisições na ordem em que chegam.

Segundo exemplo do uso de Fila: Compra de Ingressos

Vamos simular um sistema simples em que as pessoas entram em uma fila para comprar ingressos de um filme.

```
#include <iostream>
#include <queue>
#include <string>

struct Pessoa {
    std::string nome;
    int quantidadeIngressos;

    Pessoa(std::string n, int q) : nome(n), quantidadeIngressos(q) {}
};

int main() {
    std::queue<Pessoa> filaIngressos;
```

```
    // Pessoas entrando na fila para comprar ingressos
    filaIngressos.push(Pessoa("Ana", 2));
    filaIngressos.push(Pessoa("Carlos", 3));
    filaIngressos.push(Pessoa("Beatriz", 1));

    // Simulando o atendimento de compra de ingressos
    while (!filaIngressos.empty()) {
        Pessoa clienteAtual = filaIngressos.front();  // Pega a primeira pessoa da fila
        std::cout << clienteAtual.nome << " comprou "
                  << clienteAtual.quantidadeIngressos << " ingresso(s)." << std::endl;

        filaIngressos.pop();  // Remove a pessoa da fila após a compra
    }

    return 0;
}
```

Explicação:

1. **Estrutura `Pessoa`**: Cada pessoa na fila tem um nome e a quantidade de ingressos que quer comprar.
2. **Fila de Pessoas**: Criamos uma fila (`std::queue<Pessoa>`) onde as pessoas entram para comprar ingressos.
3. **Processo de Compra**: Usamos um loop `while` para atender as pessoas na ordem em que entraram na fila. Cada vez que uma pessoa compra, ela é removida da fila com `filaIngressos.pop()`.

3) Aplicando a programação orientada a objetos no sistema de compras de ingressos

Vamos refatorar o sistema de compra de ingressos usando os recursos da OOP: Classes, Atributos, métodos etc. Observe que o resultado final apresentado de ambos os sistemas é igual, mas a forma como ele é estruturado e codificado possui diferenças. Este exemplo está estruturado com os conceitos já aprendidos na orientação a objetos. Observe o código.

```
#include <iostream>
#include <queue>
#include <string>

class Pessoa {
private:
    std::string nome;
    int quantidadeIngressos;

public:
    // Construtor
    Pessoa(std::string n, int q) : nome(n), quantidadeIngressos(q) {}

    // Métodos para acessar os atributos
    std::string getNome() const {
        return nome;
    }

    int getQuantidadeIngressos() const {
        return quantidadeIngressos;
    }
};

class FilaIngressos {
private:
    std::queue<Pessoa> fila;  // Fila de pessoas

public:
    // Método para adicionar uma pessoa à fila
    void adicionarPessoa(const Pessoa& pessoa) {
        fila.push(pessoa);
    }

    // Método para processar a compra de ingressos
    void processarFila() {
        while (!fila.empty()) {
            Pessoa clienteAtual = fila.front();  // Obtém a pessoa da frente
            std::cout << clienteAtual.getNome() << " comprou "
                      << clienteAtual.getQuantidadeIngressos() << " ingresso(s)."
                      << std::endl;
```

```
            fila.pop();  // Remove a pessoa após o atendimento
        }
    }

    // Verificar se a fila está vazia
    bool filaVazia() const {
        return fila.empty();
    }
};

int main() {
    FilaIngressos filaDeIngressos;

    // Adicionando pessoas à fila de ingressos
    filaDeIngressos.adicionarPessoa(Pessoa("Ana", 2));
    filaDeIngressos.adicionarPessoa(Pessoa("Carlos", 3));
    filaDeIngressos.adicionarPessoa(Pessoa("Beatriz", 1));

    // Processando a fila de compra de ingressos
    filaDeIngressos.processarFila();

    return 0;
}
```

Explicação:

1. **Classe `Pessoa`**:
    - A Pessoa agora é uma classe com **atributos privados** (nome e quantidadeIngressos).
    - Usamos um **construtor** para inicializar esses atributos e métodos getNome() e getQuantidadeIngressos() para acessá-los externamente.

2. **Classe `FilaIngressos`**:
    - Essa classe tem um atributo fila, que é uma fila de objetos Pessoa.
    - O método adicionarPessoa() permite adicionar uma pessoa à fila.
    - O método processarFila() simula o processo de atendimento, removendo cada pessoa após a compra de ingressos.
    - O método filaVazia() verifica se a fila está vazia.

4) Considerações da aula

Nesta aula aprendemos como a estrutura Fila é aplicada em um sistema. O domínio dos conceitos de Pilha, Fila e Lista estudados em Estrutura de Dados e aplicados em Programação Orientada a objetos é uma forma de consolidar estes recursos para a manipulação de dados dentro do sistema.

Bons estudos.