UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

# Projeto de ED - Relatório Parte I

Dupla: Maria Alice Angelim Facundo e Yasmin Lima Costa
Turma 02A – Ciência da Computação
Prof. Atílio Gomes

---

## 1. Introdução

As estruturas de dados escolhidas para a implementação das políticas FCFS e SJF foram, respectivamente, uma FIFO e uma LinkedList.

O FCFS é a implementação de uma fila (ou queue) que segue a política FIFO, onde o primeiro elemento inserido é o primeiro a ser removido. A fila é implementada usando uma estrutura de lista encadeada simples. Já o SJF é a implementação de uma lista encadeada que foi manipulada a partir de uma função Sort para ordenar a lista de forma crescente.

## 2. Arquivos

O projeto é inicialmente dividido nos seguintes arquivos:

- main.cpp → atualmente, designada para testar apenas as principais funções;
- Node.h → responsável por criar os nós necessários para as duas estruturas;
- FCFS.h → esqueleto da ED com a definição de todas as funções;
- FCFS.cpp → estrutura de dados FIFO com a implementação das operações;
- SJF.h → esqueleto da ED com a definição de todas as funções;
- SJF.cpp → LinkedList modificada com função Sort.

### 2.1. Node.h

O arquivo Node.h possui uma classe que cria um nó para as estruturas de listas encadeadas.

```
class Node {
    friend class Fifo;
    friend class Iterator;
    friend class LinkedList;
```

A classe Node declara três classes como amigas: Fifo, Iterator e LinkedList, o que faz com que essas classes tenham acesso aos membros privados da classe.

```
    int data;
    Node* next;
```

Declaração dos atributos da classe, onde int data armazena o valor que o nó contém e Node* next é um ponteiro para o próximo nó na lista usado para criar uma conexão de ordem entre os nós.

```
    Node(const int& val, Node *ptrNext) {
```

O construtor da classe Node recebe dois argumentos: um valor inteiro (val) e um ponteiro para o próximo nó (ptrNext).

```
        data = val;
        next = ptrNext;
```

Ele inicializa o atributo data com o valor passado como argumento e o atributo next com o valor do ponteiro passado como argumento.

```
    ~Node() {
       std::cout << "node " << data << " foi liberado \n";
```

O destrutor, responsável por liberar a memória associada a um nó, imprime uma mensagem informando que o nó com o valor data foi liberado.

## 2.2. FCFS.cpp e FCFS.h

```
    Node* m_first {};
    Node* m_last {};
```

Cria um ponteiro do tipo Node para o início e um para o fim da lista..

```
    Fifo();
    unsigned size() const;
```

Retorna o número de elementos na fila (tamanho da fila).

```
    bool empty() const;
```

Retorna true se a fila estiver vazia, ou seja, se o tamanho da fila for zero.

```
    void push(const int& elem);
```

Insere um elemento na fila, seguindo a política FIFO, ou seja, o elemento é inserido no final da fila. Se a fila estiver vazia, um novo nó é criado e tanto o primeiro quanto o último ponteiro da fila apontam para esse nó. Mas se a fila já tiver elementos, um novo nó é criado e adicionado ao final da fila, atualizando o ponteiro "m_last".

```
    void pop();
```

Se a fila não estiver vazia,  o primeiro nó é removido e os ponteiros da fila são atualizados.

```
    int& front();
    const int& front() const;
```

A função "front()" possui duas versões, uma constante e uma não constante. Elas retornam uma referência para o primeiro elemento da fila, se a fila não estiver vazia. Caso contrário elas lançam uma exceção "std::runtime_error".

```
    int& back();
```

Assim como as funções front, com a mesma ideia, exceto que as funções back retornam uma referência para o último elemento da fila.

```
    ~Fifo();
```

O destrutor deleta todos os nós na fila chamando o destrutor do primeiro nó (m_first), o que, por sua vez, chama recursivamente os destrutores dos demais nós na lista encadeada.

```
    Fifo& operator=(const Fifo& other);
```

Este operador permite copiar o conteúdo de uma fila para outra. Ele verifica se a fila atual não é a mesma da fila que está sendo atribuída, para evitar a autoatribuição. Em seguida, ele limpa a fila atual (usando a função clear()) e copia os elementos da outra fila para a fila atual.

```
    Fifo(const Fifo& other);
```

O construtor de cópia cria uma nova fila que é uma cópia da fila passada como argumento. Ele inicializa a fila atual como vazia e, em seguida, usa o operador de atribuição para copiar os elementos da outra fila para a fila atual.

```
        Node* current;
```

Ponteiro do tipo Node para rastrear o elemento atual que o iterador está apontando na fila.

```
        Iterator(Node* node);
```

Construtor da classe Iterator. Ele recebe um ponteiro para um nó como argumento. Isso define a posição inicial do iterador na fila

```
        int& operator*();
```

Ela retorna uma referência para o valor (data) armazenado no nó atual, permitindo a leitura ou modificação desse valor.

```
        Iterator& operator++();
```

Move o iterador para o próximo nó, atualizando o membro current para apontar para o nó seguinte. A função retorna uma referência para o próprio iterador, após a atualização.

```
        bool operator!=(const Iterator& other) const;
```

Compara dois iteradores, o iterador atual e outro iterador passado como argumento. Retorna true se os dois forem diferentes.

```
    Iterator begin();
    Iterator end();
```

As funções begin() e end() permitem obter iteradores para o início e o final da fila.

```
    void clear();
```

Limpa a fila, removendo todos os elementos e liberando a memória alocada para os nós.

## 2.3. SJF.cpp e SJF.h

```
    Node* m_head {};
```

Cria um ponteiro do tipo Node para a cabeça da lista.

```
    unsigned m_size {};
```

Retorna o número de elementos na fila (tamanho da fila).

```
    void sort();
```

Ordena os elementos na lista encadeada em ordem crescente, utilizando o algoritmo de ordenação de seleção (Selection Sort) para realizar a ordenação. Os elementos são comparados uns com os outros, e os valores são trocados, se necessário, para obter a ordenação.

```
    LinkedList();
```

O construtor é definido para criar uma lista encadeada vazia com um nó cabeça inicial.

```
    LinkedList(const LinkedList& lst);
```

O construtor de cópia é definido para criar uma cópia de outra lista encadeada. Ele percorre a lista passada como argumento e cria uma nova lista com os mesmos elementos.

```
    bool empty() const;
```

Retorna true se a lista estiver vazia, ou seja, se o nó cabeça não tiver um próximo nó.

```
    int size() const;
```

Retorna o tamanho da lista, que é mantido em m_size.

```
    void clear();
```

Usado para limpar a lista, remove todos os elementos e libera a memória alocada para os nós, chamando repetidamente o método pop_front() para remover os elementos da lista.

```
    ~LinkedList();
```

Destrutor, chama o método clear() para liberar a memória alocada para a lista e, em seguida, deleta o nó cabeça.

```
    void print() const;
```
Imprime os elementos da lista encadeada.

```
    void push_front(const int& val);
```
Insere um elemento no início da lista.

```
    int& front();
    const int& front() const;

    int& back();
    const int& back() const;
```
Possuem as mesmas funcionalidades das respectivas funções do arquivo FCFS.

```
    void pop_front();
```
Remove o primeiro elemento da lista.

```
    void push_back(const int& val);
```
Insere um elemento no final da lista.

```
    void pop_back();
```
Remove o primeiro elemento da lista.

### 2.4. main.cpp

A main permite ao usuário escolher entre dois algoritmos de escalonamento, FCFS (First-Come, First-Served) ou SJF (Shortest Job First).

**Opção SJF:**

- Cinco elementos são inseridos em uma lista encadeada e ordenados em ordem crescente.
- O programa imprime a lista antes e depois da ordenação, além de mostrar seu tamanho.
- Elementos são inseridos no início e no final da lista, e o primeiro e último elementos são impressos.
- O programa remove o primeiro e o último elementos da lista e verifica se a lista está vazia.

**Opção FCFS:**

- O programa utiliza uma fila para demonstrar operações de fila com três elementos.
- Ele imprime o tamanho da fila, verifica se a fila está vazia e imprime o primeiro e último elementos.

- Remove um elemento da fila e verifica o tamanho da fila após a remoção.
- Por fim, a fila é limpa, e o programa verifica o tamanho e a condição de vazia da fila após a limpeza.

Em resumo, a main ilustra o uso das duas estruturas de dados e testa as principais operações, como inserção, remoção, acesso e verificação do estado das estruturas de dados.

## 3. Compilação

Siga o passo a passo para compilar o programa pelo VSCode:

1. Abra a pasta com os arquivos no VSCode e pressione as teclas Ctrl + Shift + ‘ para abrir o terminal;
2. Digite no terminal os seguintes comandos na mesma ordem:

**g++ -std=c++11 *.cpp -o main.exe**

**.\main.exe**