



# Pilhas e Filas

PN & JFig

2004

PN&JF 2004

```
class Elemento{  
public:  
    float valor;  
    Elemento* esquerda;  
    Elemento* direita;  
    Elemento* pai;
```

```
    Elemento(float valor){  
        this->valor = valor;
```

```
    /*  
     * Quando não assumimos quem não  
     * é filho de sua esquerda e a sua  
     * esquerda = direita = NULL;  
     */  
};
```

```
void InserirElemento(Elemento* e){  
    if(e == NULL) return;  
    e->esquerda = e->direita = e->pai = NULL;  
    e->valor << endl;
```



# Pilhas

```
class Elemento{
public:
    float valor;
    Elemento* esquerda;
    Elemento* direita;
    Elemento* pai;

    Elemento(float valor){
        this->valor = valor;

        /*
        Quando criamos um nó assumimos que não
        tem nenhum nó à sua esquerda e à sua
        direita
        */
        esquerda = direita = NULL;
    };

    void escreva(Elemento* e){
        if(e == NULL) return;
        escreva(e->esquerda);
        cout << e->valor << endl;
        escreva(e->direita);
    }
};
```

# Pilhas - definição

- Uma pilha é uma estrutura linear de dados que pode ser acessada unicamente de uma das extremidades.

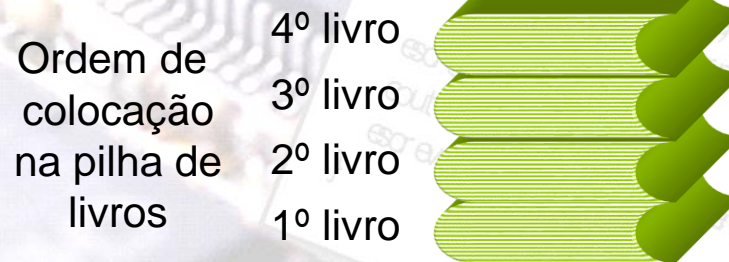
```
class Elemento{
public:
    float valor;
    Elemento* esquerda;
    Elemento* direita;
    Elemento* pai;

    Elemento(float valor) {
        // Quando criamos um nó assumimos que não
        // fica nenhum nó à sua esquerda e à sua
        // direita.
        esquerda = direita = NULL;
    };

    void escreva(Elemento* e) {
        if (e == NULL) return;
        escreva(e->esquerda);
        cout << e->valor << endl;
        escreva(e->direita);
    }
};
```

# Pilhas

- Podemos ver uma pilha como um conjunto de livros, colocados uns em cima dos outros.





# Pilhas

- Para retirar os livros começamos pelo último livro colocado na pilha.

Ordem de  
colocação  
na pilha de  
livros

4º livro

3º livro

2º livro

1º livro



1º livro retirado

# Pilhas

- Por este motivo, a pilha é designada por uma estrutura do tipo LIFO (**L**ast-**I**n-**F**irst-**O**ut).



# Pilhas

- As operações normalmente definidas para manipulação de uma pilha são:
  - **clear()** – eliminar todos os elementos da pilha;
  - **isEmpty()** – verificar se a pilha está vazia;
  - **push(elemento)** – colocar um elemento no topo da pilha;
  - **pop()** – retirar o elemento mais alto da pilha;
  - **topEl()** – devolve o elemento da mais alto da pilha, sem o eliminar;



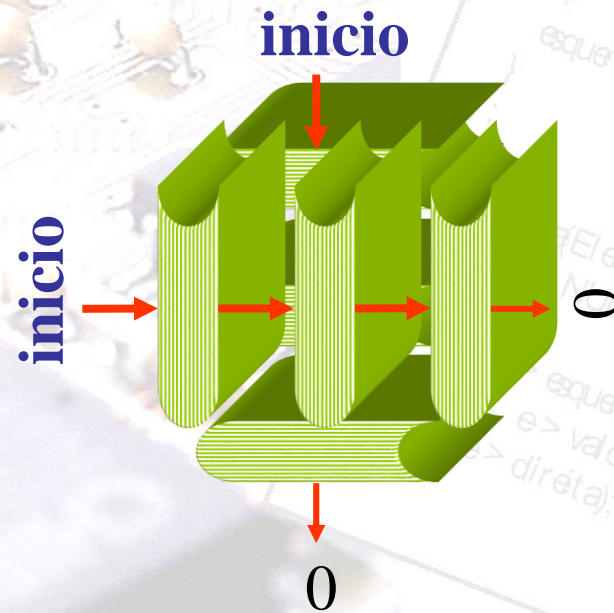
# Pilhas

- Partindo da definição existente para a lista simples e utilizando as vantagens da utilização de classes, facilmente chegamos à implementação da **class Pilha**.



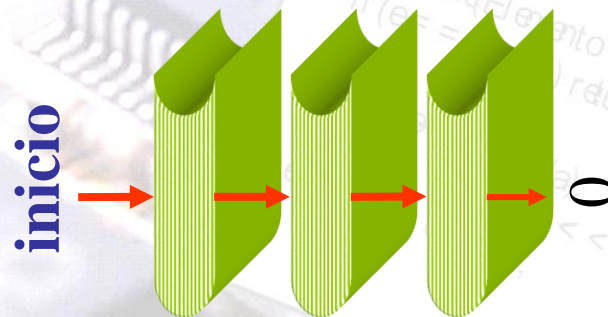
# Pilhas

- A pilha não é mais que uma implementação específica da lista.



# Pilhas: implementação

- No caso específico da pilha, vamos utilizar os métodos *inserirInicio()* e *eliminarInicio()*, implementados na classe **Lista**, para a manipulação da pilha.



# Pilhas: implementação

```
class Lista{  
    private:  
        Elemento * inicio;  
    public:  
        Lista(){ inicio = 0; }  
        Lista(int valor)  
        void inserirInicio(int valor);  
        void inserirFim(int valor);  
        int eliminaInicio();  
        int eliminaFim();  
        string verTudo();  
        bool vazia();  
};
```

```
class TPilha : protected Lista {  
    public:  
        TPilha() { Lista(); }  
        void push(int e);  
        int pop();  
        string verTudo();  
        bool isEmpty();  
        void clear();  
};
```



# Pilhas: class TPilha

```
void TPilha::push(int e){  
    inserirInicio(e);  
}
```

```
int TPilha::pop(){  
    return eliminarInicio();  
}
```

```
bool isEmpty(){ return vazia(); }
```

```
void clear() { limpar(); }
```



A close-up photograph of several integrated circuit (IC) chips on a printed circuit board (PCB). The chips are dark grey or black with gold-colored pins. On the surface of the chips, there are tiny, colorful figures of people, some standing and some walking, which are part of the chip's packaging design.

# Filas

```
class Elemento{
public:
    float valor;
    Elemento* esquerda;
    Elemento* direita;
    Elemento* pai;

    Elemento(float valor){
        this->valor = valor;
    }

    /*
    Quando criamos um nó assumimos que não
    há nenhum nó à sua esquerda e à sua
    direita.
    */
    esquerda = direita = NULL;
};

void escreve(Elemento* e){
    if(e == NULL) return;
    escreve(e->esquerda);
    cout << e->valor << endl;
    escreve(e->direita);
}
```

# Filas: definição

- Fila é uma linha de espera que cresce adicionando elementos no seu final e que diminui retirando elementos do seu início.

# Filas

- Para melhor entendermos as filas, consideremos uma fila de pessoas. Cada vez que chega uma pessoa à fila, esta fica atrás das outras que já lá estavam. A primeira pessoa a sair da fila é sempre a que chegou primeiro, depois a que chegou em segundo lugar e assim sucessivamente.



# Filas

- Uma fila é designada por uma estrutura do tipo FIFO (**F**irst-**I**n-**F**irst-**O**ut). Ou seja, o primeiro elemento a entrar na fila é o primeiro a sair.



# Filas: operações frequentes

- As operações mais frequentes para manipulação de uma fila são:
  - **clear()** – eliminar todos os elementos da fila;
  - **isEmpty()** – verificar se a fila está vazia;
  - **enqueue(elemento)** – colocar um elemento no final da fila;
  - **dequeue()** – retira o primeiro elemento da fila;

# Filas: implementação

- Da mesma forma que as pilhas, e tirando partido das características das classes, vamos utilizar a **class Lista** para implementar a **class TFila**.
- As filas não são mais que uma implementação específica das listas.

# Pilhas: implementação

```
class Lista{  
    private:  
        Elemento * inicio;  
    public:  
        Lista(){ inicio = 0; }  
        Lista(int valor)  
        void inserirInicio(int valor);  
        void inserirFim(int valor);  
        int eliminaInicio();  
        int eliminaFim();  
        string verTudo();  
        bool vazia();  
};
```

```
class TFila : protected Lista {  
    public:  
        TFila() { Lista(); }  
        void enqueue(int e);  
        int dequeue();  
        string verTudo();  
        bool isEmpty();  
        void clear();  
};
```



# Filas: class TFila

```
void TFila::enqueue(int e){  
    inserirFim(e);  
}
```

```
int TFila::dequeue(){  
    return eliminalInicio();  
};
```

```
bool isEmpty(){ return vazia() };  
void clear(){ limpa() };
```