

- 
- 
- 
- 
- 
- 
- 

# Pilhas e Filas

**Algoritmos e Estruturas de Dados**

2020/2021



FEUP

- 
- 
- 
- 
- 
- 
- 
-

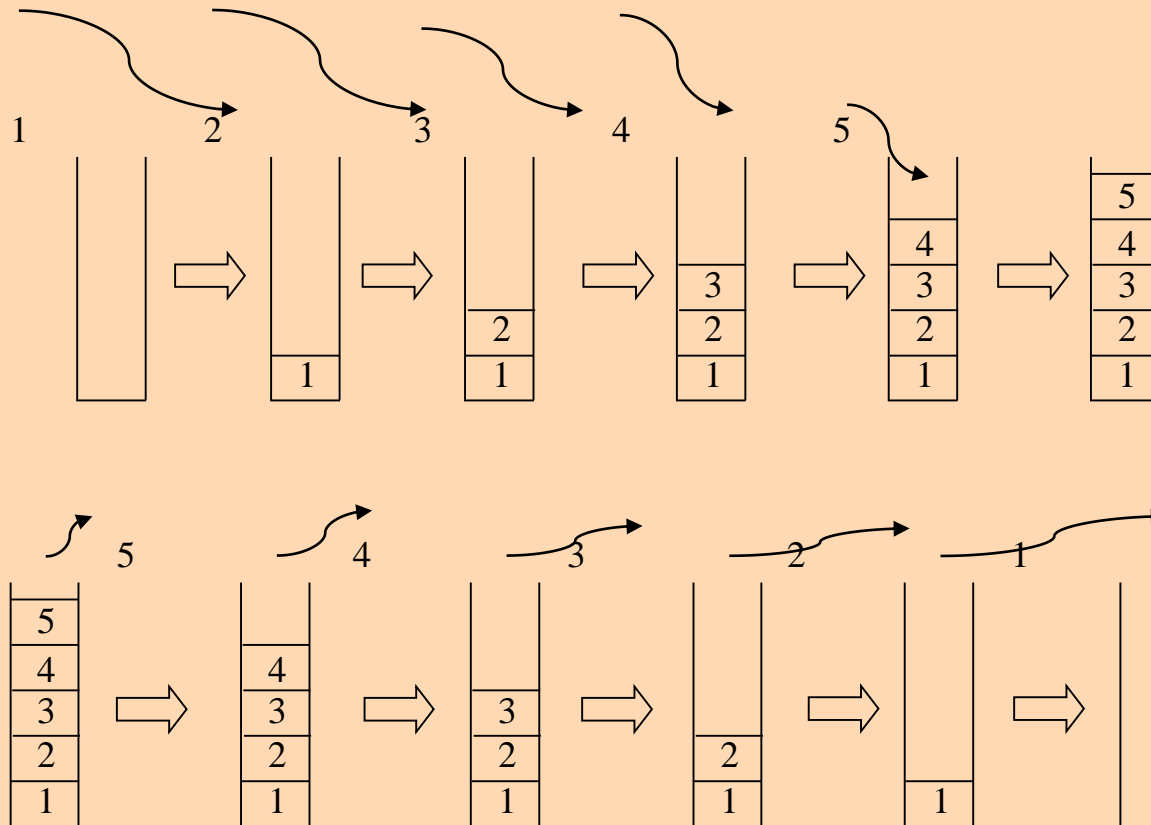
# Pilhas

- Pilha
  - estrutura de dados linear em que a inserção e a remoção de elementos de uma sequência se faz pela mesma extremidade, designada por **topo** da pilha
  - uma pilha pode ser considerada como uma *restrição* de lista
  - porque é uma estrutura de dados mais simples que a lista, é possível obter implementações mais eficazes
  - o conceito de iterador **não se aplica** a esta estrutura de dados
  - a pilha é uma estrutura do tipo LIFO (*Last-In-First-Out*)
  - Operações mais usuais:
    - criar uma pilha vazia
    - adicionar/remover um elemento a uma pilha
    - verificar qual o último elemento colocado na pilha



# Pilhas

- Operações com pilhas



# Pilhas (Standard Template Library - STL)

- class *stack*<T>
- Alguns métodos:
  - stack & *operator* =(const stack &)
  - bool *empty*() const
  - size\_type *size*() const
  - T & *top*()
  - void *push*(const T &)
  - void *pop*()
  - bool *operator* ==(const stack &, const stack &)
  - bool *operator* <(const stack &, const stack &)

(<http://www.sgi.com/tech/stl/stack.html>)

# Aplicações de pilhas

## Notação RPN (*Reverse Polish Notation*)

- expressões matemáticas em que os operadores surgem a seguir aos operandos (notação pósfixa)
- vantagem: não requer parêntesis nem regras de precedência

## Notação infixa

- os operadores binários surgem entre os operandos

Notação infixa:  $2 * (4 + 5) / 3$

Notação RPN:  $2\ 4\ 5\ +\ *\ 3\ /$



# Aplicações de pilhas

## Avaliação de expressões RPN : *algoritmo*

1. Processar sequencialmente os elementos da expressão.

Para cada elemento:

- 1.1 Se o elemento for um número (operando), colocá-lo na pilha
- 1.2. Se o elemento for um operador
  - 1.2.1. Retirar os dois elementos do topo da pilha
  - 1.2.2. Processar os elementos de acordo com o operador
  - 1.2.3. Colocar o resultado na pilha
1. Retirar o (único) elemento da pilha. É o resultado

# Aplicação de pilhas: avaliação de expressões

```
float calcOp(float v1, float v2, char op)
{
    switch(op) {
        case '+' : return v1+v2;
        case '-' : return v1-v2;
        case '*' : return v1*v2;
        case '/' : return v1/v2;
        default: throw OperacaoInvalida();
    }
}
```

# Aplicação de pilhas: avaliação de expressões

```
float avaliaRPN(string expressao)
// simplificação: números de 1 dígito apenas
{
    stack<float> pilha;
    for ( int i=0; i<expressao.length(); i++ ) {
        if ( expressao[i]>='0' && expressao[i]<='9')    // é número
            pilha.push(expressao[i]-48);
        else {
            float num1 = pilha.top();
            pilha.pop();
            float num2 = pilha.top();
            pilha.pop();
            float res = calcOp(num1, num2, expressao[i]);
            pilha.push(res);
        }
    }
    float res = pilha.top();  pilha.pop();
    return res;
}
```





# Filas

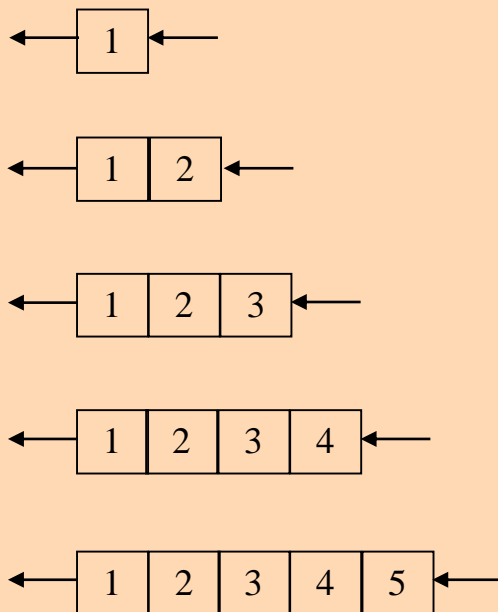
- Fila
  - estrutura de dados linear em que a inserção e a remoção de elementos de uma sequência se faz por extremidades opostas, geralmente designadas por **cabeça** e **cauda** da fila
  - uma fila pode ser considerada como uma *restrição* de lista
  - porque é uma estrutura de dados mais simples que a lista, é possível obter implementações mais eficazes
  - o conceito de iterador **não se aplica** a esta estrutura de dados
  - a fila é uma estrutura do tipo FIFO (*First-In-First-Out*)
  - Operações mais usuais:
    - criar uma fila vazia
    - adicionar/remover um elemento a uma fila
    - verificar qual o elemento da cabeça da fila (elemento mais antigo)



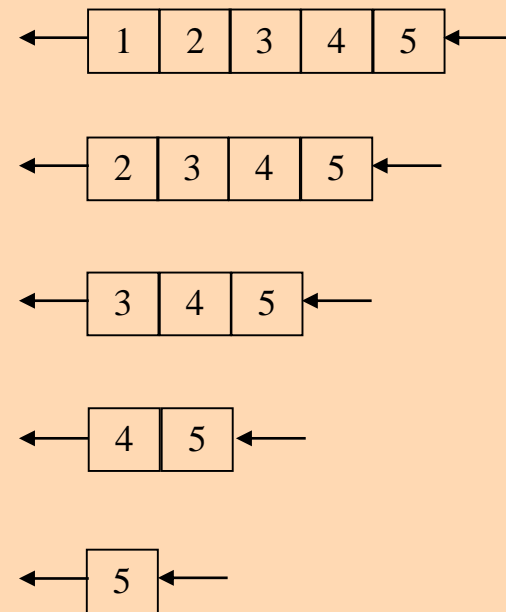
# Filas

- Operações com filas

*Inserir: 1, 2, 3, 4, 5*



*Remover: 1, 2, 3, 4, 5*



# Filas (Standard Template Library - STL)

- class *queue*<*T*>
- Alguns métodos:
  - queue & *operator* =(const queue &)
  - bool *empty*() const
  - size\_type *size*() const
  - T & *front*()
  - T & *back*()
  - void *push*(const T &)
  - void *pop*()
  - bool *operator* ==(const queue &, const queue &)
  - bool *operator* <(const queue &, const queue &)



# Aplicações de filas

- Pretende-se implementar uma classe para gerir a fila de impressão de uma impressora de rede. Considere que uma impressora de rede, quando recebe um ficheiro para impressão, adiciona o mesmo numa fila de espera obedecendo a um regime FIFO.

```
class Document
{
    string nome;
    string owner;
    int tamanho;
    int nFolhas;
public:
    Document(string n, string o, int t, int f): nome(n),
                                                owner(o), tamanho(t), nFolhas(f){}
    friend class Printer;
};
```



# Aplicações de filas

```
class Printer
{
    queue<Document> filaDeImpressao;
    int maxMemoria;
    int folhasRestantes;
public:
    Printer(int m, int f): maxMemoria(m), folhasRestantes(f) {}
    bool adicionaDocumento(Document doc);
    Document imprimeDocumento();
    void trocaToner(int numeroFolhas);
    void listaDocumentos(ostream& os);
    void excluiDocumento(string n, string o);
};
```

# Aplicações de filas

```
// Adiciona um novo documento à fila de impressão. Se o tamanho do documento  
// a adicionar é superior à memória disponível, o documento é ignorado.
```

```
bool Printer::adicionaDocumento(Document doc)  
{  
    //calcula memória consumida  
    int memoriaConsumida = 0;  
    queue<Document> temp(filaDeImpressao);  
    while(!temp.empty()) {  
        memoriaConsumida += temp.front().tamanho;  
        temp.pop();  
    }  
    bool res=false;  
    //adiciona documento  
    if( (maxMemoria - memoriaConsumida) >= doc.tamanho ) {  
        filaDeImpressao.push(doc);  
        res = true;  
    }  
    cout << filaDeImpressao.size() << endl;  
    return res;  
}
```



# Aplicações de filas

```
// Imprime o próximo documento na lista de espera. Se o documento for impresso na
// totalidade, sai da fila. Se não houver toner suficiente para imprimir todas as
// páginas, imprime as que forem possíveis, deixando o restante para ser impresso
// quando o toner for trocado e lança uma exceção do tipo Erro.
```

```
Document Printer::imprimeDocumento()
{
    Document d = filaDeImpressao.front();
    if( folhasRestantes >= d.nFolhas ) {
        folhasRestantes -= d.nFolhas;
        filaDeImpressao.pop();
    }
    else {
        d.nFolhas -= folhasRestantes;
        filaDeImpressao.front() = d;
        folhasRestantes = 0;
        throw Erro();
    }
    return d;
}
```



# Aplicações de filas

```
// Retira da fila de impressão o documento de nome n, pertencente ao owner o.

void Printer::excluiDocumento(string n, string o)
{
    queue<Document> temp;
    //apaga todos os documentos com nome n e owner o
    while(!filaDeImpressao.empty()) {
        if( (filaDeImpressao.front().nome == n) &&
            (filaDeImpressao.front().owner == o) )
            filaDeImpressao.pop();
        else {
            temp.push( filaDeImpressao.front() );
            filaDeImpressao.pop();
        }
    }
    filaDeImpressao = temp;
}
```

