ESTRUTURAS DE DADOS

Fila (Lista Encadeada)

Roteiro

- **Estrutura do Nó**
- **Tipo Abstrato de Dados**
- **Detalhes de Implementação**
- **Aplicações da Estrutura**

Estrutura do Nó

Mesma da aula passada

```
typedef char ItemType;
/*
 Estrutura usada para guardar
 a informação e o endereço do
 próximo elemento.
*/
struct NodeType
{
  ItemType info;
  NodeType* next;
};
```

Tipo Abstrato de Dados

```
class Queue
{
 public:
  Queue(); // Constructor
  ~Queue(); // Destrutor
  bool isEmpty() const;
  bool isFull() const;
  void print() const;

  void enqueue(ItemType);
  ItemType dequeue();
 private:
  NodeType* front;
  NodeType* rear;
};
```

Não mudaremos a interface pública

Mudaremos a implementação interna

Detalhes de Implementação

Implementaremos uma fila como lista encadeada.

Dois ponteiros front e rear apontarão para o início e final da fila, respectivamente.

Queremos que inserções e remoções ocorram em tempo constante. Em outras palavras, independem do número de elementos na estrutura.

Implementando as ideias da aula passada.

- **Construtor e Destrutor**

```
Queue::Queue()
{
  front = NULL;
  rear = NULL;
}

Queue::~Queue()
{
  NodeType* tempPtr;
  while (front != NULL) {
      tempPtr = front;
      front = front->next;
      delete tempPtr;
    }
  rear = NULL;
}
```

Implementando as ideias da aula passada.

- **Verificação de cheio ou vazio.**

```
bool Queue::isFull() const
{
  NodeType* location;
  try {
      location = new NodeType;
      delete location;
      return false;
  } catch(std::bad_alloc exception) {
    return true;
  }
}

bool Queue::isEmpty() const
{
  return (front == NULL);
}
```

Implementando as ideias da aula passada.

- **Inserindo elementos**

```
void Queue::enqueue(ItemType newItem)
{
  if (!isFull()) {
    NodeType* newNode;
    newNode = new NodeType;
    newNode->info = newItem;
    newNode->next = NULL;
    if (rear == NULL)
      front = newNode;
    else
      rear->next = newNode;
    rear = newNode;
  } else {
    throw "Queue is already full!";
  }
}
```

Implementando as ideias da aula passada.

- **Removendo elementos**

```
ItemType Queue::dequeue()
{
  if (!isEmpty()) {
    NodeType* tempPtr;
    tempPtr = front;
    ItemType item = front->info;
    front = front->next;
    if (front == NULL)
      rear = NULL;
    delete tempPtr;
    return item;
  } else {
    throw "Queue is empty!";
  }
}
```

Implementando as ideias da aula passada.

- **Imprimindo a lista na saída padrão**

```
void Queue::print() const
{
  NodeType* tempPtr = front;
  while (tempPtr != NULL)
    {
      cout << tempPtr->info;
      tempPtr = tempPtr->next;
    }
  cout << endl;
}
```

Usar a estrutura se assemelha ao anterior:

```
char character;
Queue queue;
char queueChar;
cout << "Enter a string; press return." << endl;
cin.get(character);
while (character != '\n' and !queue.isFull())
  {
    queue.enqueue(character);
    cin.get(character);
  }
while (!queue.isEmpty())
  {
    queueChar = queue.dequeue();
    cout << queueChar;
  }
cout << endl;
```

Aplicações da Estrutura

Uma fila é uma estrutura bastante útil, principalmente quando precisamos garantir que processos acessarão recursos compartilhados de uma maneira justa.

- **Documentos enviados para a impressão.**
- **Troca de mensagens entre processos em um Sistema Operacional.**

- **Para exercitar o uso de uma fila, trataremos do problema de verificar se uma string é um palíndrome.**

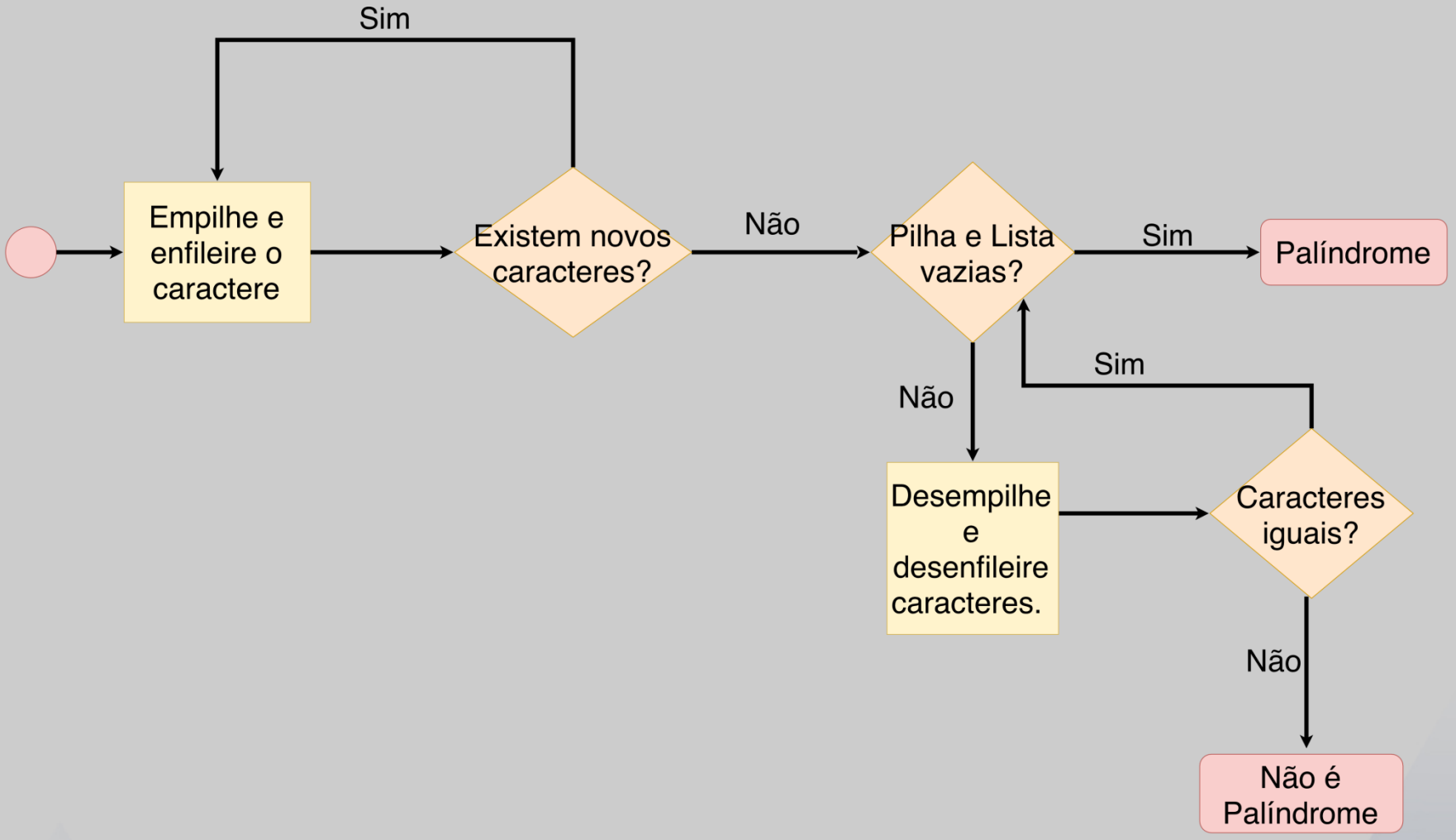
Uma *string* é um palíndrome se:

- **Contém zero ou mais caracteres.**
- **A leitura da string de trás para frente e de frente para trás é a mesma.**

Exemplos:

- **ovo Bem formada**
- **missa e assim Bem formada**
- **arara Bem formada**
- **ulisses Mal formada**
- **abracadabra Mal formada**

Fluxograma da Aplicação:



```
int main() {
  bool palindrome = true;

  char character;
  char stackChar;
  char queueChar;

  Stack stack;
  Queue queue;
  cout << "Adicione uma string." << endl;
  cin.get(character);
  while (character != '\n') {
      stack.push(character);
      queue.enqueue(character);
      cin.get(character);
    }
```

```
  while (palindrome && !queue.isEmpty())
    {
      stackChar = stack.pop();
      queueChar = queue.dequeue();
      if (stackChar != queueChar)
        palindrome = false;
    }
  if (palindrome)
    cout << "String é Palindrome" << endl;
  else
    cout << "String não é palindrome" << endl;
  return 0;
}
```

ESTRUTURAS DE DADOS

Fila (Lista Encadeada)