Filas

Prof. Denio Duarte
Prof. Geomar Schreiner

Fila (queue)

- Uma fila é uma estrutura de dados que é uma especialização da lista simplesmente encadeada
- Implementa o conceito de FIFO: first-in first-out, ou seja, o primeiro elemento que entrou é o primeiro a sair.
  - Não é possível inserir no meio ou no início. SEMPRE NO FIM
- Uma fila tem as seguintes propriedades:
  - Podemos realizar duas operações básicas:
    - **Inserir** um novo item na fila (put)
    - **Remover** um item da fila (get)
  - A operação de remoção sempre **remove o item que está há mais tempo** na fila (FIFO)

Fila (queue)

- O sistema operacional utiliza este conceito para:
  - Controlar a fila de impressão
  - Controla a execução do processos
- Dois conceitos baseados em fila:
  - Fila com prioridade em que a inserção obedece a prioridade do item a ser inserido
  - Fila circular: o tamanha do fila é fixo e é necessário controlar o início e fim da fila. Quando são iguais, a fila está vazia, quando o fim chegou na capacidade da fila, esta está cheia

Exemplo de funcionamento

Entrada:

E S * T R D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila

Fila

Fila vazia

Saída:

Exemplo de funcionamento

Entrada:

S * T R D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila

Saída:

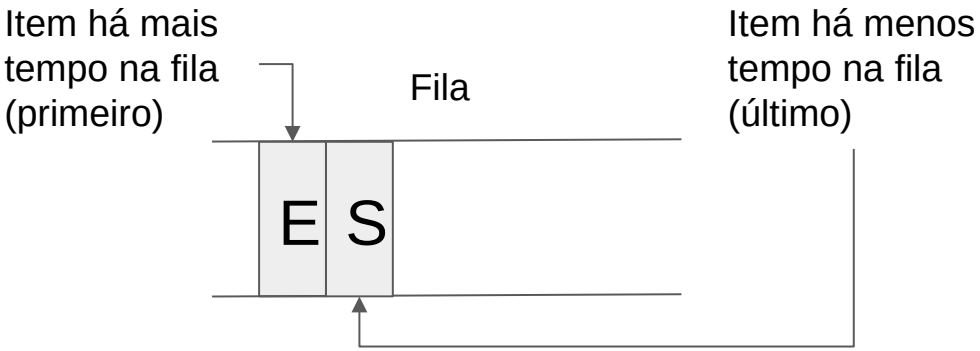
Exemplo de funcionamento

Entrada:

* T R D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

Exemplo de funcionamento

Entrada:

T R D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila

Saída:

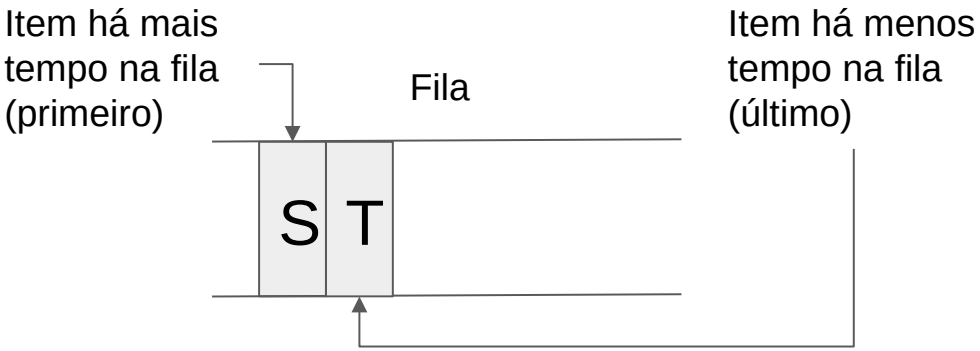
Exemplo de funcionamento

Entrada:

R D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

E

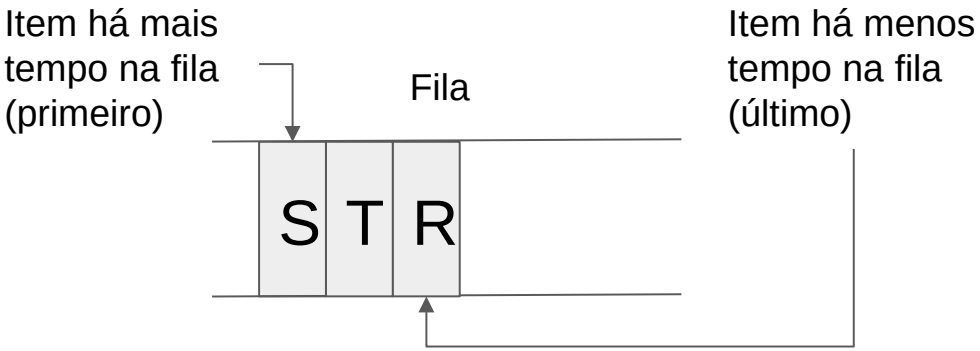
Exemplo de funcionamento

Entrada:

D * A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

E

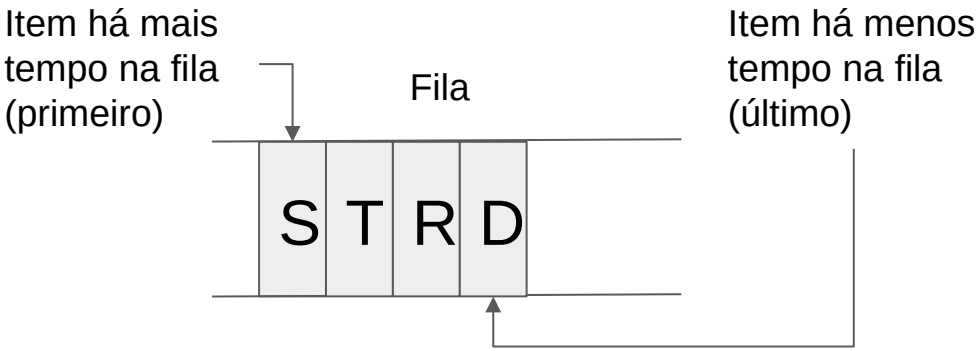
Exemplo de funcionamento

Entrada:

* A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

E

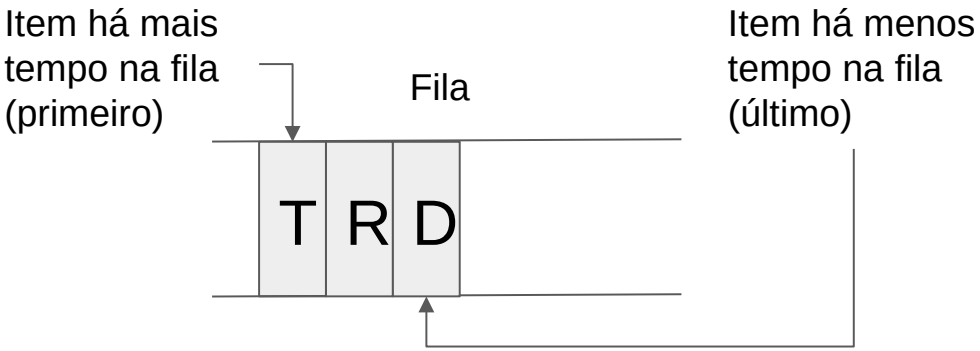
Exemplo de funcionamento

Entrada:

A ...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

E S

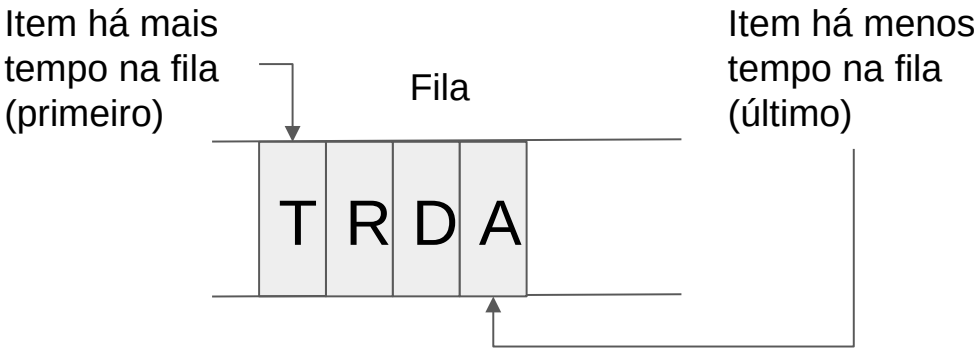
Exemplo de funcionamento

Entrada:

...

Processamento:

- Se é uma letra: insere na fila
- Se é um *: remove da fila



Saída:

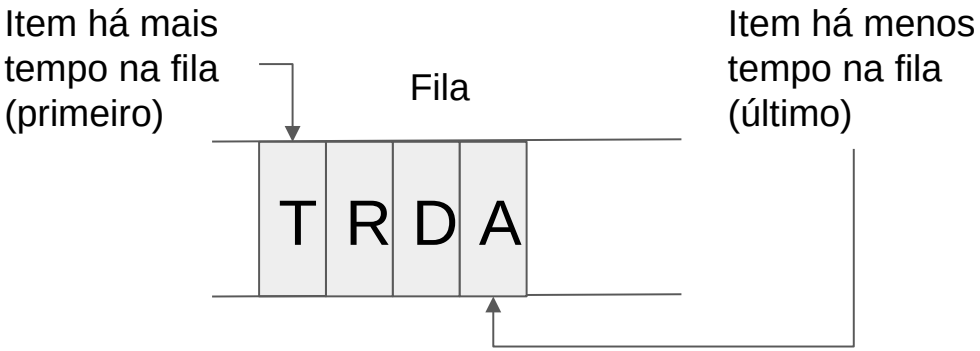
E S

Exemplo de funcionamento

Entrada:

O processamento continuaria...

...



Saída:

E S

Implementação

- Uma fila poder ser implementada de duas maneiras: usando um vetor ou usando uma lista encadeada simples
- Cada opção de implementação possui vantagens e desvantagens (as mesmas das opções de implementação de uma pilha)
- Usando um vetor
  - Desvantagem: É necessário definir um tamanho máximo da fila; uso ineficiente da memória total alocada
  - Vantagem: Inserção e remoção de itens não requerem alocação e liberação de memória
- Usando uma lista encadeada simples
  - Desvantagem: Inserção e remoção de itens requerem alocação e liberação de memória
  - Vantagem: Uso mais eficiente da memória total alocada

Implementação - lista encadeada simples

- Podemos declarar os seguintes tipos:

```
typedef int Item;
```

```
typedef struct elemFila {
    Item item;
    struct elemFila *proximo;
} ElemFila;
```

```
typedef struct {
    ElemFila *primeiro;
    ElemFila *ultimo;
} Fila;
```

Implementação - lista encadeada simples

- Podemos declarar os seguintes tipos:

```
typedef int Item;
```

```
typedef struct elemFila {
    Item item;
    struct elemFila *proximo;
} ElemFila;
```

```
typedef struct {
    ElemFila *primeiro;
    ElemFila *ultimo;
} Fila;
```

```
Fila *fila;
```

Implementação - lista encadeada simples

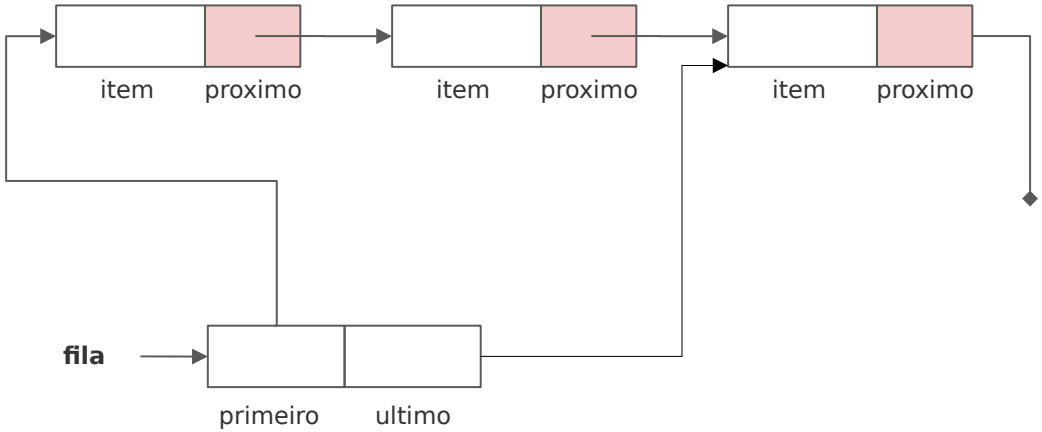
- Podemos declarar os seguintes tipos:

```
typedef int Item;
```

```
typedef struct elemFila {
    Item item;
    struct elemFila *proximo;
} ElemFila;
```

```
typedef struct {
    ElemFila *primeiro;
    ElemFila *ultimo;
} Fila;
```

```
Fila *fila;
```

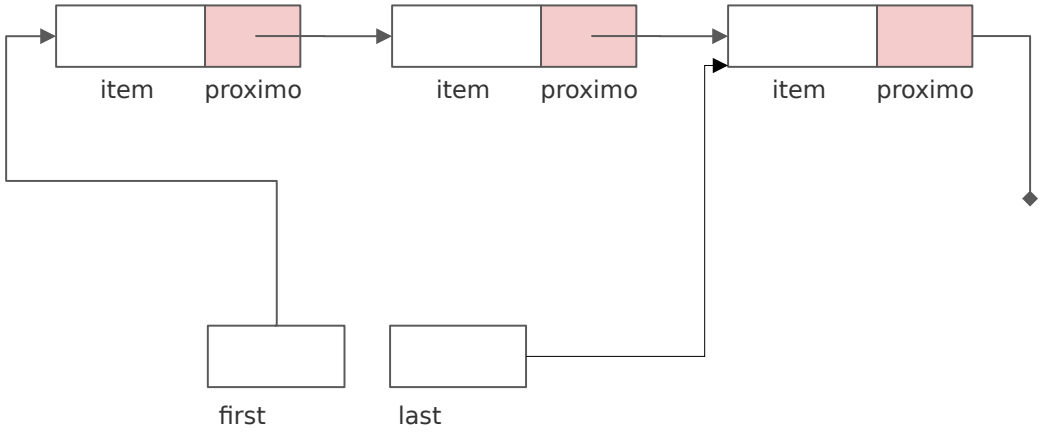


Implementação - lista encadeada simples

- Ou utilizando dois ponteiros para controla início e fim.

```
typedef int Item;
```

```
typedef struct elemFila {
    Item item;
    struct elemFila *proximo;
} ElemFila;
```




```
Fila *fila, *first, *last;
```

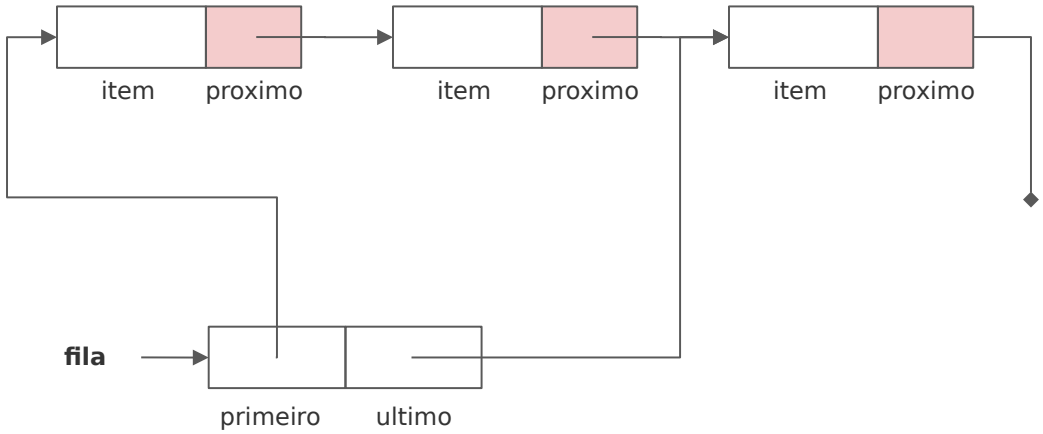
Implementação - lista encadeada simples

- Operações:

```
void insereFila(Fila *fila, Item item)

void removeFila(Fila *fila, Item *item)

void inicializaFila(Fila *fila)

int filaVazia(Fila *fila)

void liberaFila(Fila *fila)
```

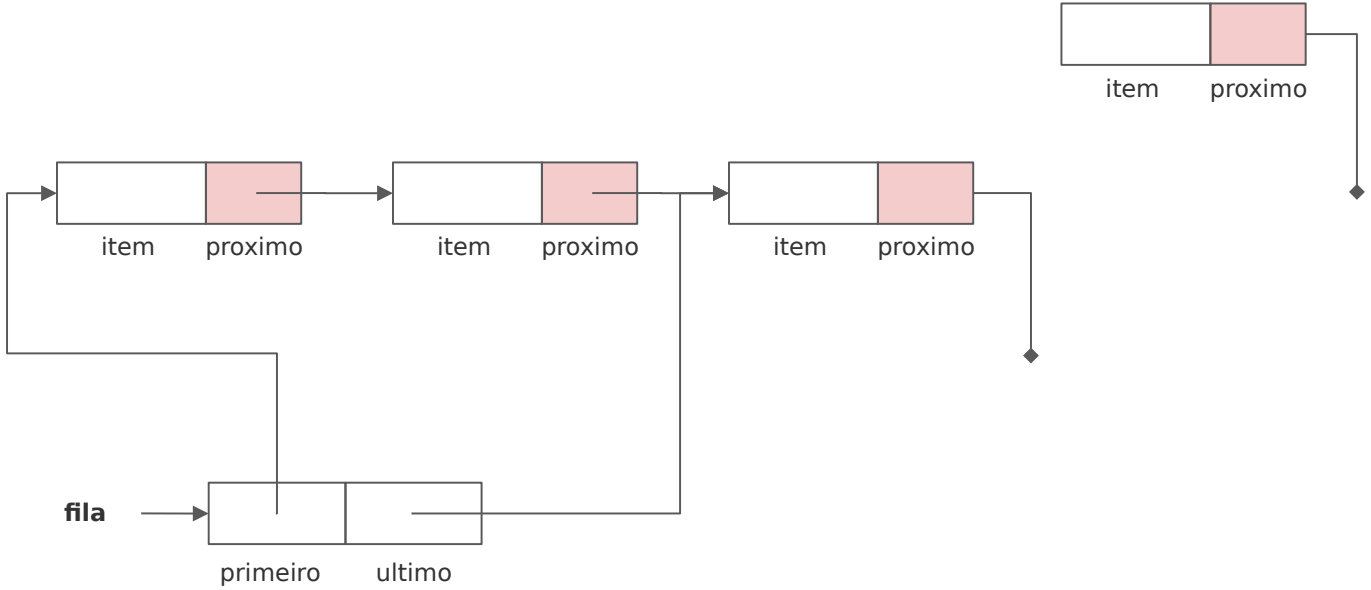
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



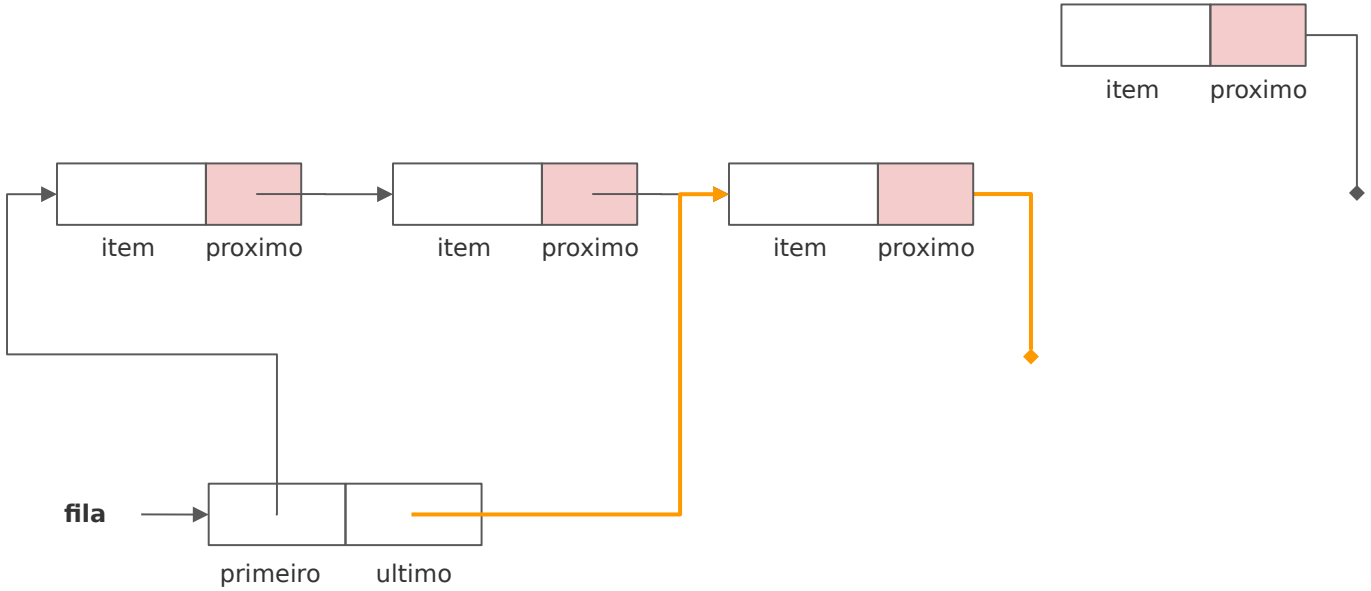
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



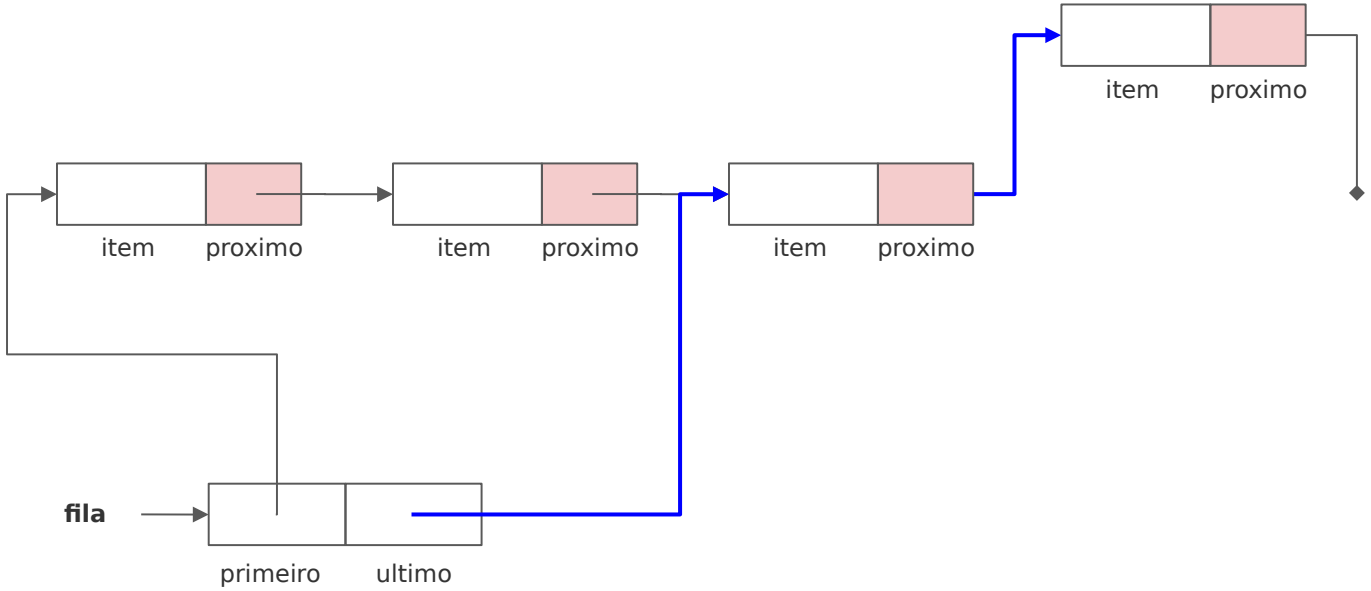
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



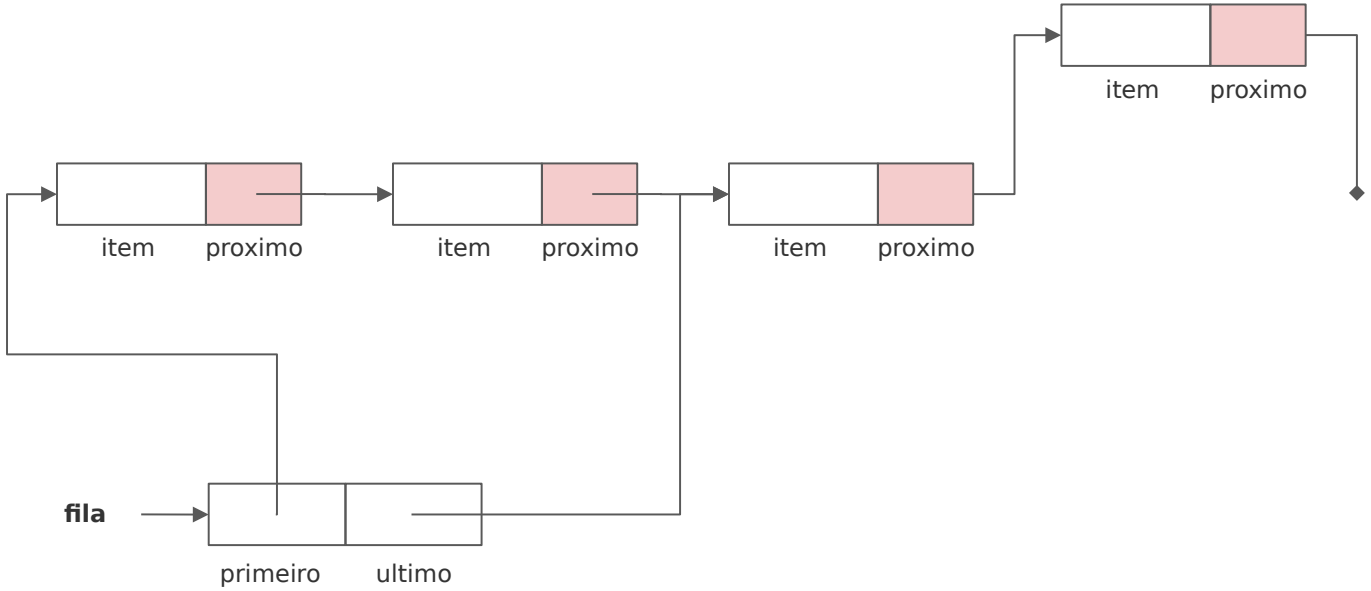
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



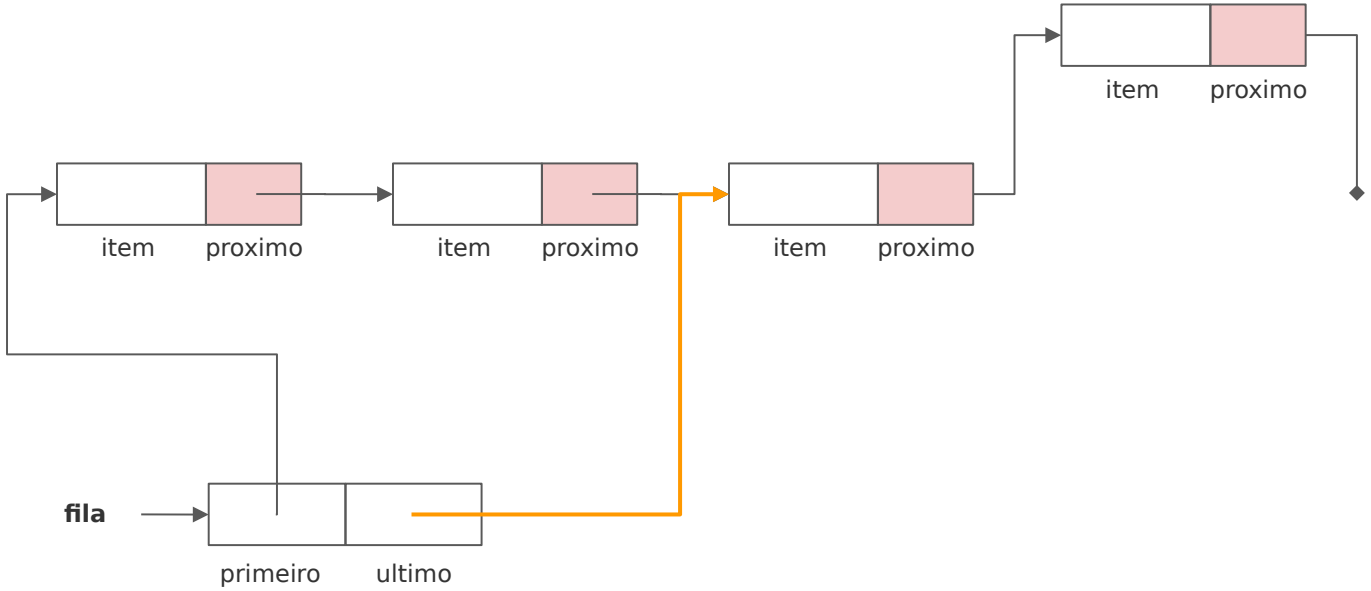
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



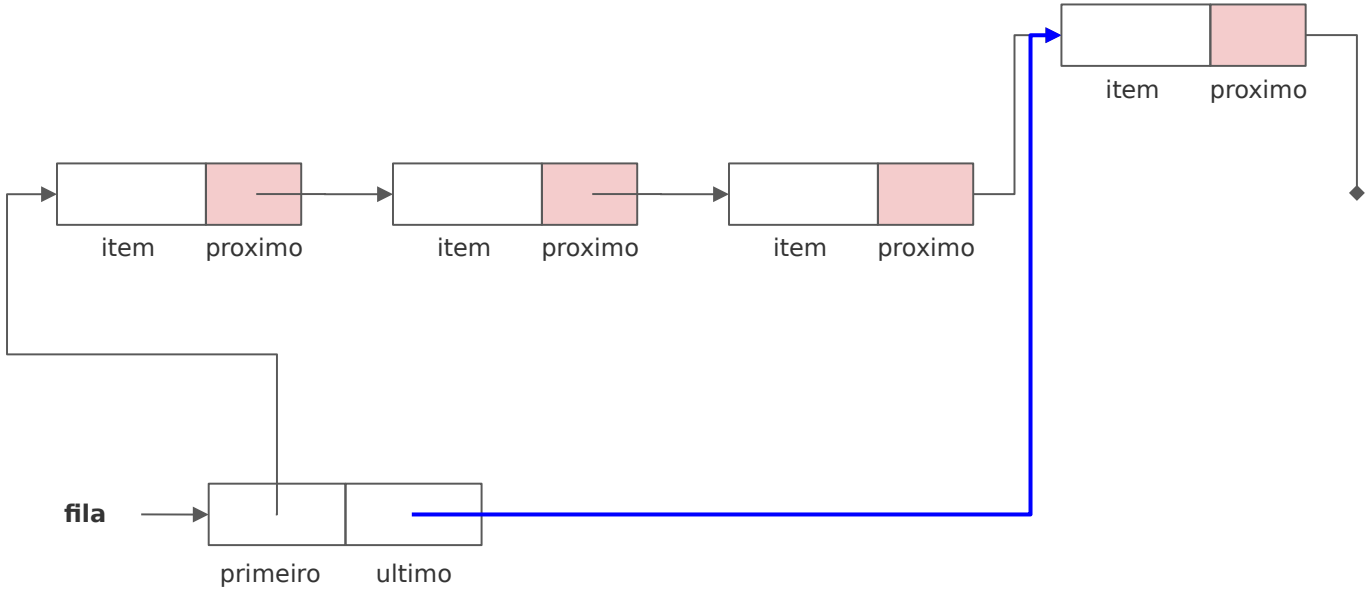
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



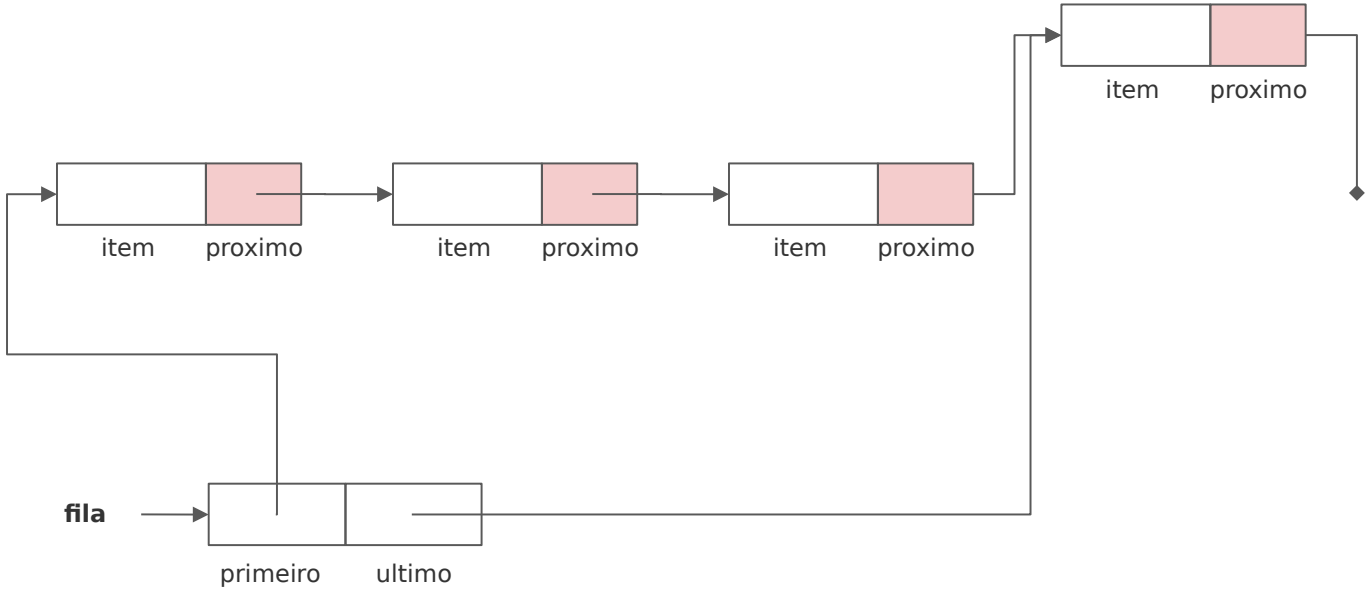
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:



Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:

```
void insereFila(Fila *fila, Item item) {
   ElemFila *aux;

   // Cria um novo elemento da lista encadeada que representa a fila e
   // armazena neste novo elemento o item a ser inserido na fila
   aux = malloc(sizeof(ElemFila));
   aux->item = item;
   aux->proximo = NULL;

   // Continua no proximo slide
```

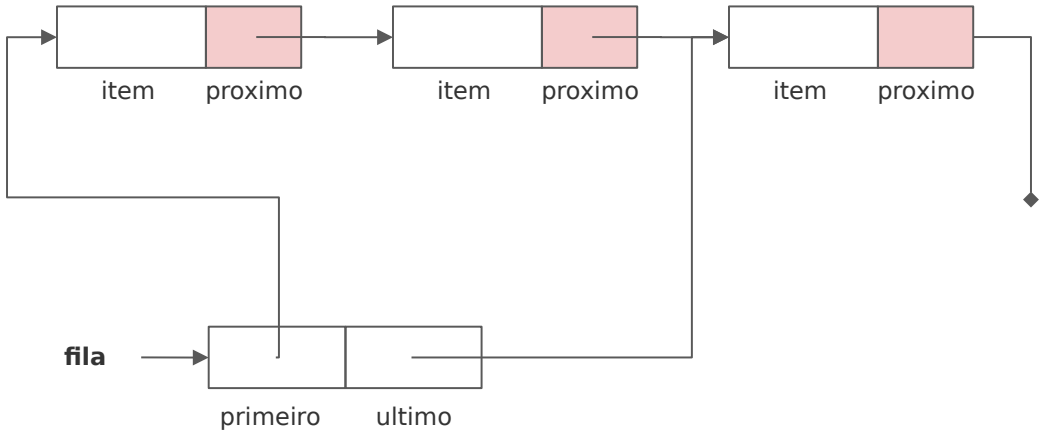
Implementação - lista encadeada simples

- Operação de inserir um novo elemento na fila:

```
    // Continuacao do slide anterior

    // Insere o novo elemento no fim da lista encadeada que representa a
    // fila
    if (fila->primeiro == NULL) { // Se a fila esta vazia
        fila->primeiro = aux;
        fila->ultimo = aux;
    }
    else { // Se a fila nao esta vazia
        fila->ultimo->proximo = aux;
        fila->ultimo = aux;
    }
}
```

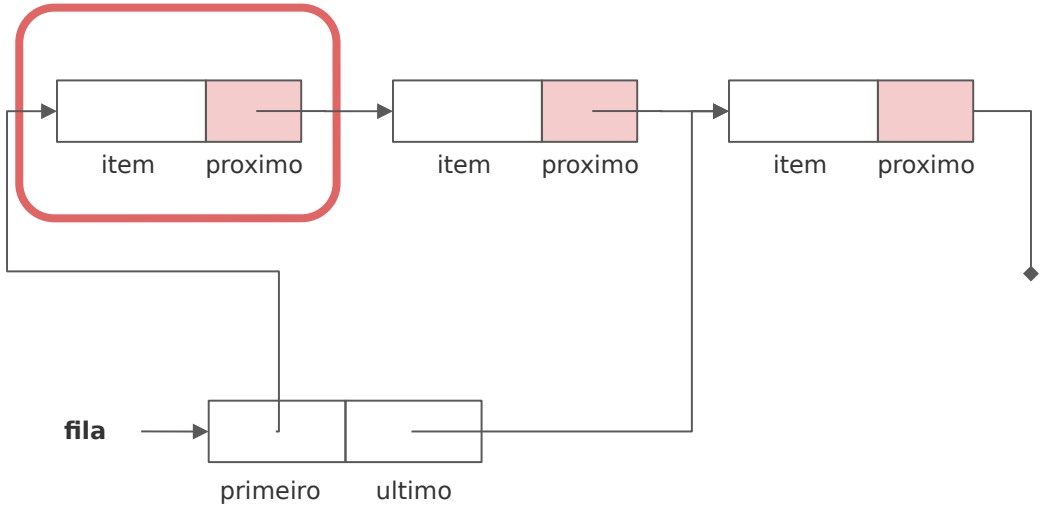
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



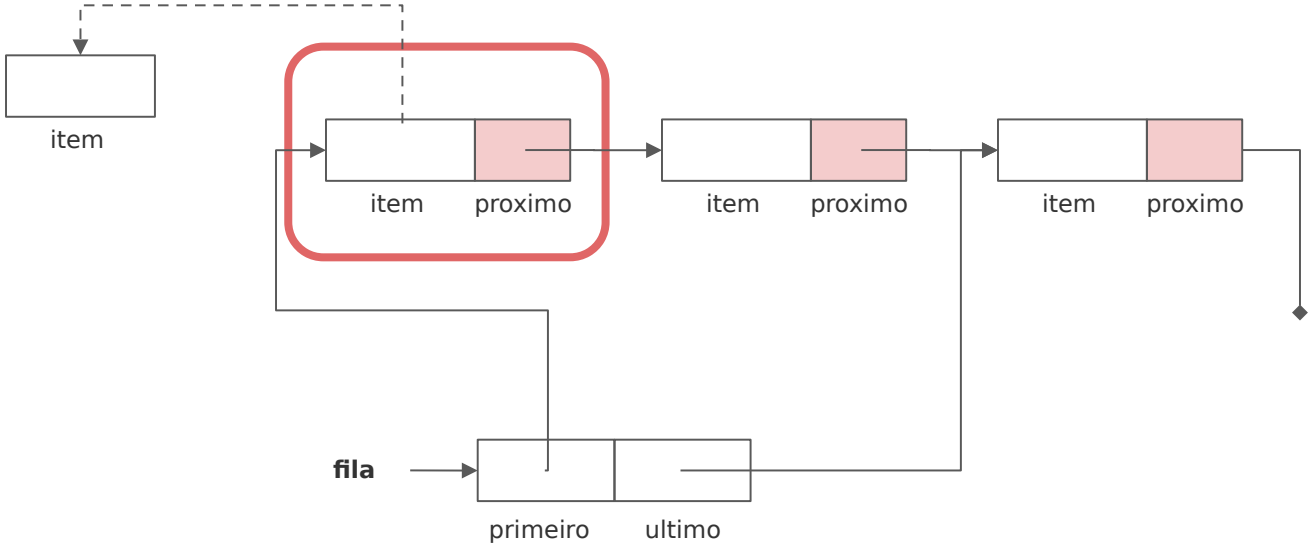
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



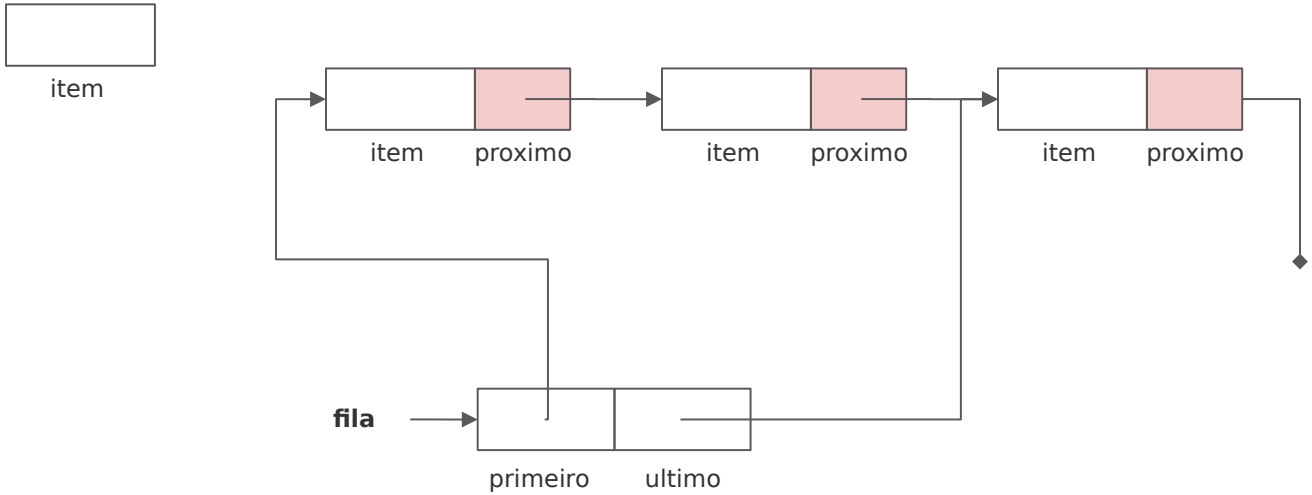
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



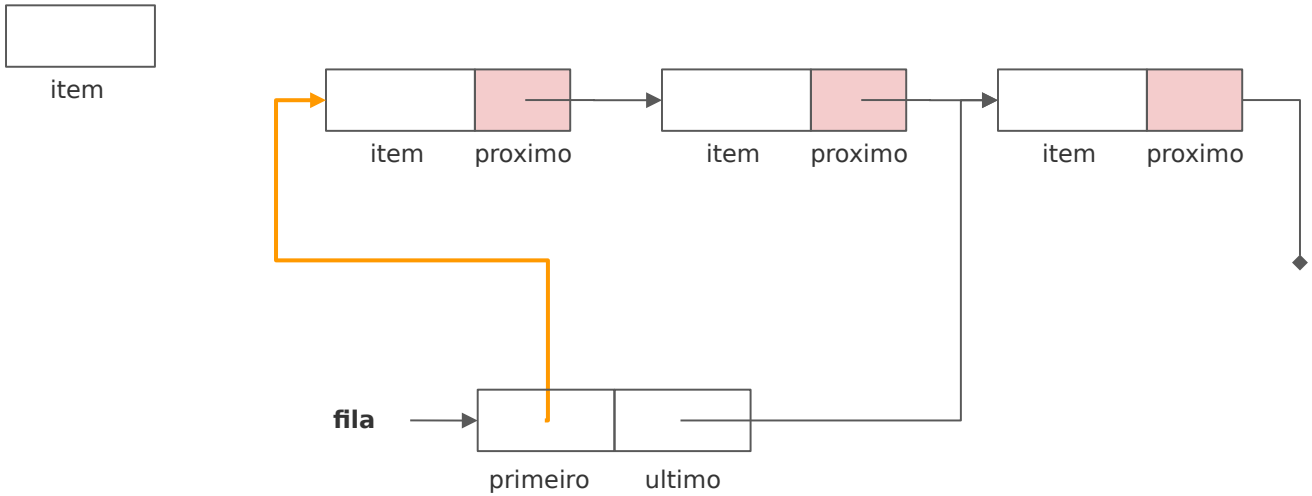
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



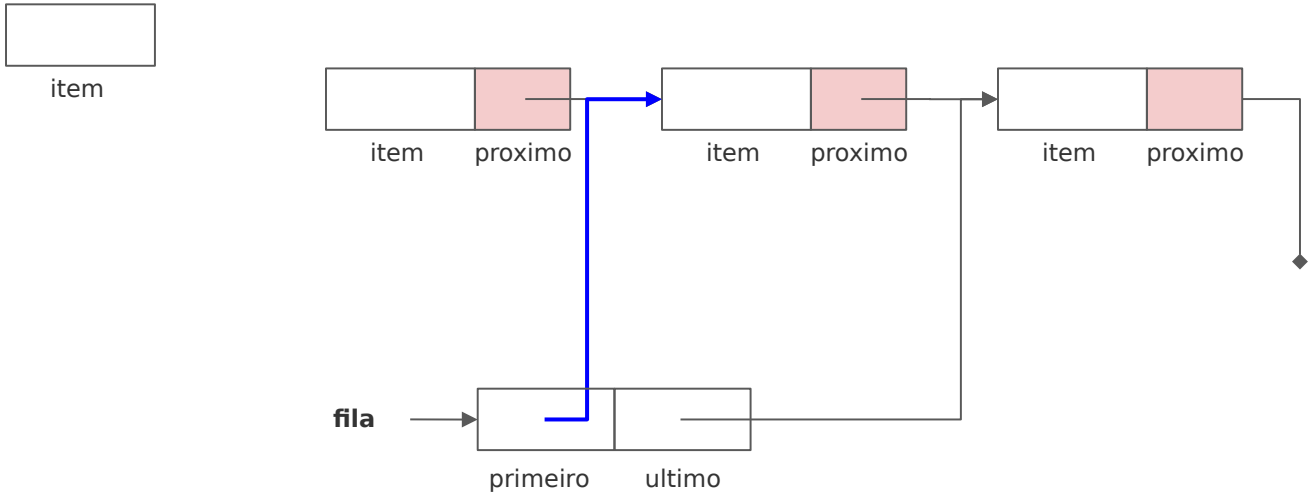
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



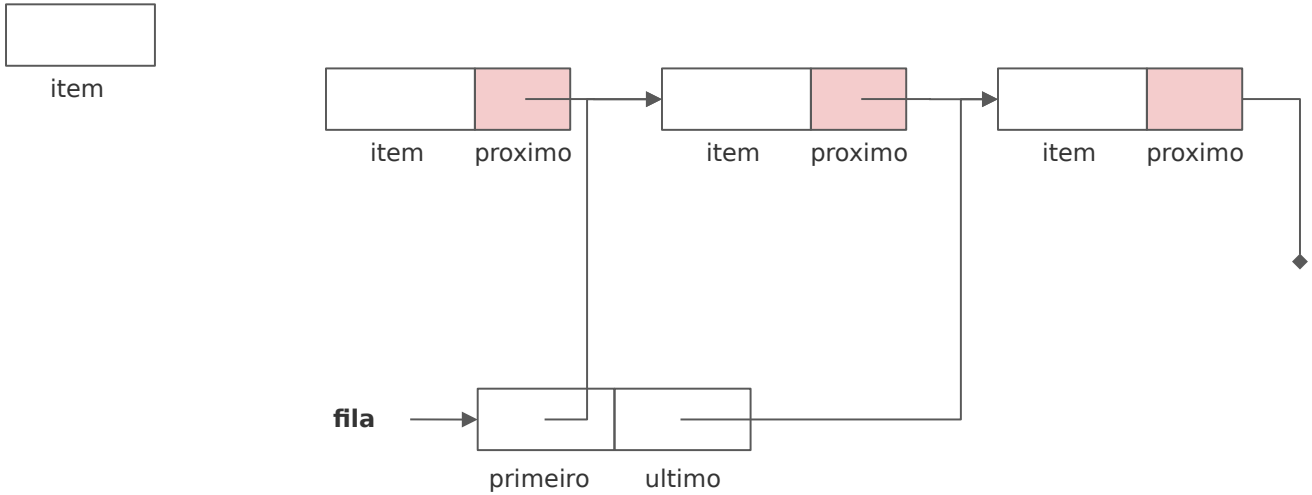
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



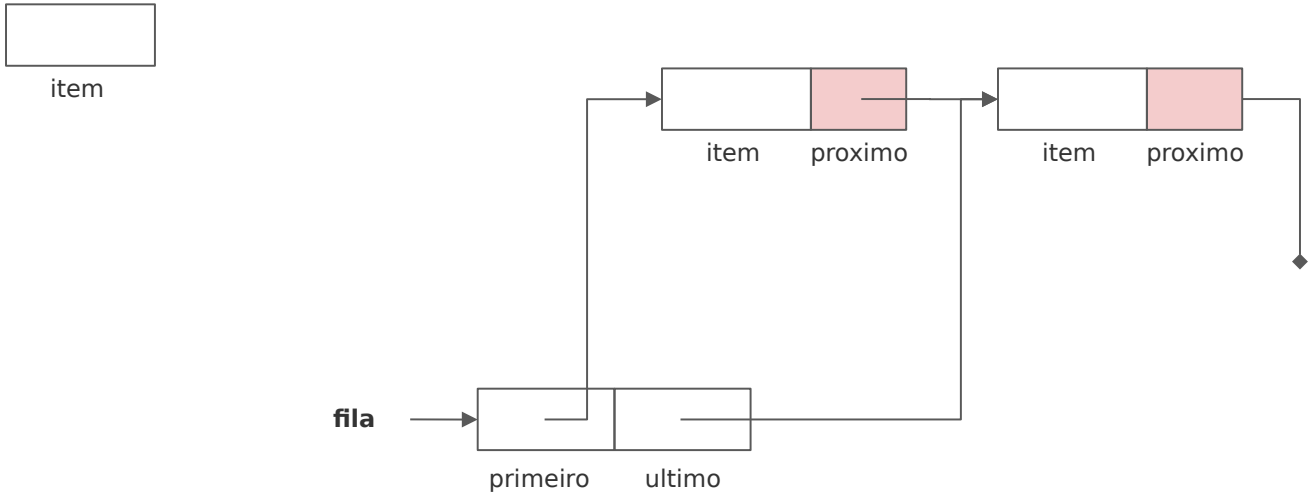
Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



Implementação - lista encadeada simples

- Operação de remover um elemento da fila:



Implementação - lista encadeada simples

- Operação de remover um elemento da fila:

```
void removeFila(Fila *fila, Item *item) {
    ElemFila *aux;

    // Verificar se a fila esta vazia!

    // Armazena o item a ser removido da fila
    *item = fila->primeiro->item; // ATENCAO: Depende da definicao do tipo do
                                  // item

    // Continua no proximo slide
```

Implementação - lista encadeada simples

- Operação de remover um elemento da fila:

```
    // Continuacao do slide anterior

    // Armazena o primeiro elemento da lista encadeada que representa a fila e
    // remove este primeiro elemento da lista
    aux = fila->primeiro;
    if (fila->primeiro == fila->ultimo) {
        fila->primeiro = NULL;
        fila->ultimo = NULL;
    }
    else {
        fila->primeiro = fila->primeiro->proximo;
    }

    // Libera a memoria alocada para o elemento removido
    free(aux);
}
```

Implementação - lista encadeada simples

- Operação de inicializar a fila:

```
void inicializaFila(Fila *fila) {
   fila->primeiro = NULL;
   fila->ultimo = NULL;
}
```

Implementação - lista encadeada simples

- Operação de testar se a fila está vazia:

```
int filaVazia(Fila *fila) {
   return (fila->primeiro == NULL);
}
```

Implementação - lista encadeada simples

- Operação de destruir a fila (liberar a memória alocada para a fila):

```
void liberaFila(Fila *fila) {
   ElemFila *aux;

   while (fila->primeiro != NULL) {
      // Armazena o primeiro elemento da lista encadeada que representa a
      // fila e remove este primeiro elemento da lista
      aux = fila->primeiro;
      fila->primeiro = fila->primeiro->proximo;

      // Libera a memoria alocada para o elemento removido
      free(aux);
   }
   fila->ultimo = NULL;
}
```

Implementação - lista encadeada simples

- Usando a fila:

```
int main() {
    Fila fila;
    Item item;

    inicializaFila(&fila);

    for (int i = 0; i < 10; i++) {
        item = i;

        printf("Inserindo na fila o item %d.\n", item);
        insereFila(&fila, item);
    }

    // Continua no proximo slide
```

Implementação - lista encadeada simples

- Usando a fila:

```
    // Continuacao do slide anterior

    while (filaVazia(&fila) == 0) { // Enquanto a fila nao esta vazia
        removeFila(&fila, &item);
        printf("Item %d removido da fila.\n", item);
    }

    liberaFila(&fila); // Sem efeito se a fila ja esta vazia

    return 0;
}
```

Exercícios

1. Faça um programa que receba uma string, caracter por caracter
   - Cada caracter é colocado em uma fila
   - No fim da entrada, esvazie a pilha imprimindo os caracteres armazenados

Referências

- Os exercícios desta apresentação são baseados no seguinte material: Pfenning, F., Platzer, A., Simmons, R., Lecture 9 - Stacks and Queues, Lecture Notes, Carnegie Mellon University, 2020. (https://www.cs.cmu.edu/~15122/handouts/09-stackqueue.pdf)