



Estrutura de Dados

Filas

Prof. Thiago Caproni Tavares ¹ Paulo Muniz de Ávila ²

¹thiago.tavares@ifsuldeminas.edu.br

²paulo.avila@ifsuldeminas.edu.br

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

- Utiliza uma política FIFO (*first in first out*): o primeiro elemento inserido será o primeiro a ser removido.
- Cada elemento armazena um ou vários dados (estrutura homogênea ou heterogênea) e um ponteiro para o próximo elemento, permitindo o encadeamento e mantendo a estrutura linear;
- **Operações:** inserir, consultar, remover e esvaziar;
- Possui um ponteiro INÍCIO (remoções) e um FIM (inserções);
 - As operações ocorrem nas duas extremidades da estrutura.

1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

1 Introdução

2 Operações

- Inicialização

- Inserindo na Lista

- Remoções

3 Análise da Complexidade

Inicialização

1º Operação:
Inicializa a lista

```
void inicializar (Fila *fila){  
    fila->inicio = NULL;  
    fila->fim = NULL;  
    fila->tam = 0;  
}
```

Fila	
tam	
lixo	
inicio	fim
NULL	lixo

Inicialização

1º Operação:
Inicializa a lista

```
void inicializar (Fila *fila){  
    fila->inicio = NULL;  
    fila->fim = NULL;  
    fila->tam = 0;  
}
```

Fila	
tam	
lixo	
inicio	fim
NULL	NULL

Inicialização

1º Operação:
Inicializa a lista

```
void inicializar (Fila *fila){  
    fila->inicio = NULL;  
    fila->fim = NULL;  
    fila->tam = 0;  
}
```

Fila	
tam	
0	
inicio	fim
NULL	NULL

1 Introdução

2 Operações

- Inicialização
- **Inserindo na Lista**
- Remoções

3 Análise da Complexidade

Inicialização

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```

Fila	
tam	
0	
inicio	fim
NULL	NULL

500	
dado	prox
lixo	lixo
novo	
500	

Inicialização

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```

Fila	
tam	
0	
inicio	fim
NULL	NULL

500	
dado	prox
lixo	lixo
novo	
500	

Inicialização

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```

Fila	
tam	
0	
inicio	fim
NULL	NULL

500	
dado	prox
3	lixo
novo	
500	

Inicialização

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inicialização

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

2ª Operação:
Inserção do #3

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

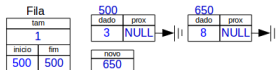
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

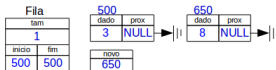
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

3ª Operação:
Inserção do #8

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

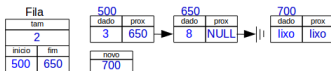
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

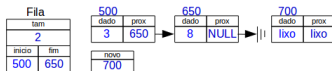
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

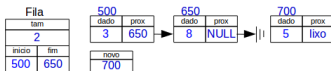
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

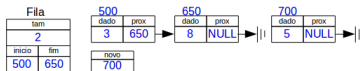
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

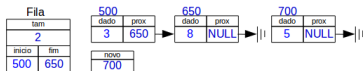
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

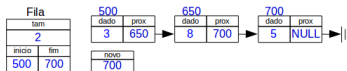
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

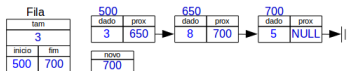
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

4ª Operação:
Inserção do #5

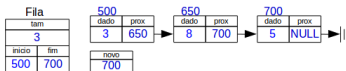
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

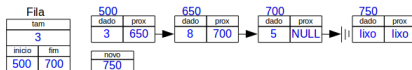
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

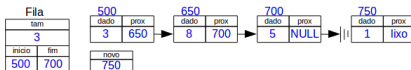
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

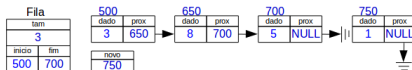
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

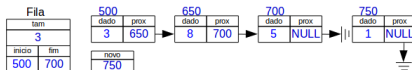
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

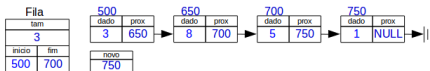
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

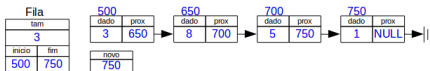
```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



Inserindo no Início da Lista

5ª Operação:
Inserção do #1

```
int inserir(Fila *fila, int dado){
    cel *novo = malloc(sizeof(cel));

    if(novo == NULL)
        return 0;

    novo->dado = dado;
    novo->prox = NULL;

    if(fila->inicio == NULL){
        fila->inicio = novo;
        fila->fim = novo;
    }else{
        fila->fim->prox = novo;
        fila->fim = novo;
    }

    fila->tam++;
    return 1;
}
```



1 Introdução

2 Operações

- Inicialização
- Inserindo na Lista
- Remoções

3 Análise da Complexidade

Removendo Elementos

6ª Operação:
Remoção da Fila

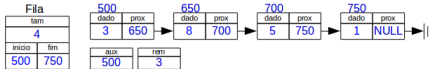
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

6ª Operação:
Remoção da Fila

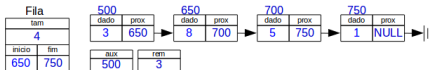
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

6ª Operação:
Remoção da Fila

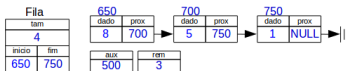
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

6ª Operação:
Remoção da Fila

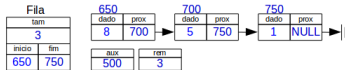
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

6ª Operação:
Remoção da Fila

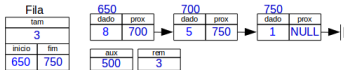
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

6ª Operação:
Remoção da Fila

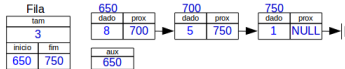
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

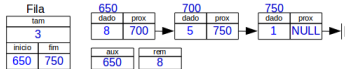
```
int remover(Fila *fila){
    cel *aux = fila->inicio;
    int rem = aux->dado;
    fila->inicio = fila->inicio->prox;
    free(aux);
    fila->tam--;
    return rem;
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

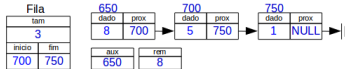
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

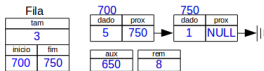
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

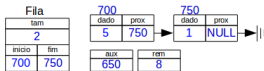
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

7ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

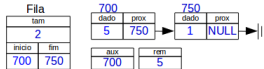
```
int remover(Fila *fila){
    cel *aux = fila->inicio;
    int rem = aux->dado;
    fila->inicio = fila->inicio->prox;
    free(aux);
    fila->tam--;
    return rem;
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

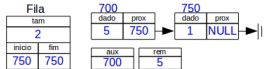
```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



Removendo Elementos

8ª Operação:
Remoção da Fila

```
int remover(Fila *fila){  
    cel *aux = fila->inicio;  
    int rem = aux->dado;  
    fila->inicio = fila->inicio->prox;  
    free(aux);  
    fila->tam--;  
    return rem;  
}
```



1 Introdução

2 Operações


- Inicialização
- Inserindo na Lista
- Remoções


3 Análise da Complexidade

- A inserção sempre realiza operações básicas, para atualizar o INÍCIO e FIM da fila;
- O mesmo ocorre no caso da remoção para atualizar o INÍCIO;
 - São operações de tempo constante e gastam $O(1)$.
- Consultar toda a fila percorre os elementos armazenados. Uma fila contém n elementos:
 - logo o tempo de execução é $O(n)$.
- A operação de esvaziamento da fila remove todos os elementos:
 - logo o tempo de execução é $O(n)$.


Obrigado pela atenção!!!
thiago.tavares@ifsuldeminas.edu.br





 ASCENCIO, A.; CAMPOS, E. de. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java*. Pearson Prentice Hall, 2008. ISBN 9788576051480. Disponível em: <<https://books.google.com.br/books?id=p-mTPgAACAAJ>>.


 C: A Reference Manual. Pearson Education, 2007. ISBN 9788131714409. Disponível em: <<https://books.google.com.br/books?id=Wt2NEypdGNIC>>.


 DAMAS, L. *LINGUAGEM C*. LTC. ISBN 9788521615194. Disponível em: <<https://books.google.com.br/books?id=22-vPgAACAAJ>>.

 FEOFILOFF, P. *Algoritmos Em Linguagem C*. CAMPUS - RJ, 2009. ISBN 9788535232493. Disponível em: <<http://books.google.com.br/books?id=LfUQai78VQgC>>.

 KERNIGHAN, B.; RITCHIE, D. *C: a linguagem de programação padrão ANSI*. Campus, 1989. ISBN 9788570015860. Disponível em: <<https://books.google.com.br/books?id=aVWrQwAACAAJ>>.

 LOPES, A.; GARCIA, G. *Introdução à programação: 500 algoritmos resolvidos*. Campus, 2002. ISBN 9788535210194. Disponível em: <<https://books.google.com.br/books?id=Rd-LPgAACAAJ>>.

 MIZRAHI, V. *Treinamento em linguagem C*. Pearson Prentice Hall, 2008. ISBN 9788576051916. Disponível em: <<https://books.google.com.br/books?id=7xt7PgAACAAJ>>.

 SCHILDT, H.; MAYER, R. *C completo e total*. Makron, 1997. ISBN 9788534605953. Disponível em: <<https://books.google.com.br/books?id=Pbl0AAAACAAJ>>.