

MC-202

Filas e Pilhas

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2024-09-18 10:57

Filas

- Uma impressora é compartilhada em um laboratório
- Alunos enviam documentos quase ao mesmo tempo



Filas

- Uma impressora é compartilhada em um laboratório
- Alunos enviam documentos quase ao mesmo tempo



Como gerenciar a lista de tarefas de impressão?

Fila

Fila:

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo:



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()




Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()




Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Desenfileira()**



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Desenfileira()**




Fila

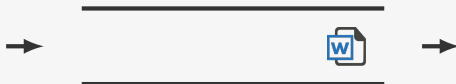
Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()




Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()




Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Enfileira**()



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Desenfileira()**



Fila

Fila:

- Remove primeiro objetos **inseridos há mais tempo**
- **FIFO** (*first-in first-out*): primeiro a entrar é primeiro a sair

Operações:

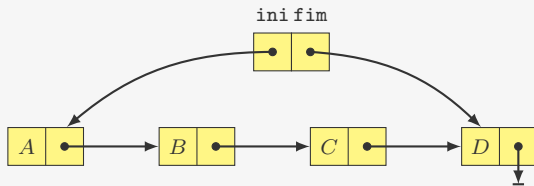
- **Enfileira** (*queue*): adiciona item no “fim”
- **Desenfileira** (*dequeue*): remove item do “início”

Exemplo: **Desenfileira()**

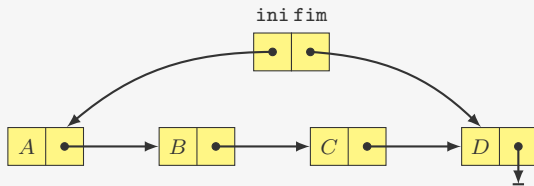


Fila: implementação com lista ligada

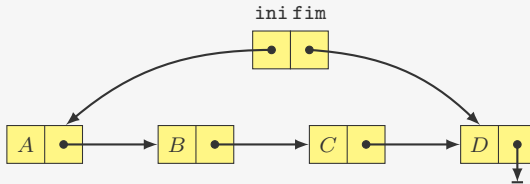
Fila: implementação com lista ligada



Fila: implementação com lista ligada

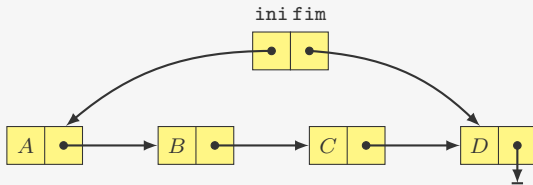


Fila: implementação com lista ligada



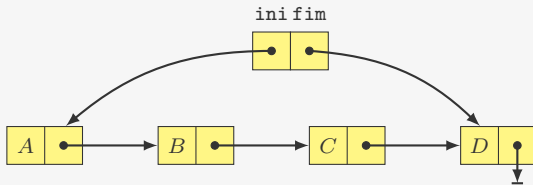
```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     p_no ini, fim;  
5 };
```

Fila: implementação com lista ligada



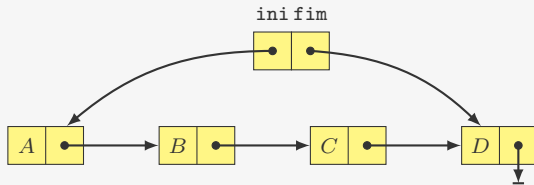
```
1 p_fila criar_fila() {
```

Fila: implementação com lista ligada



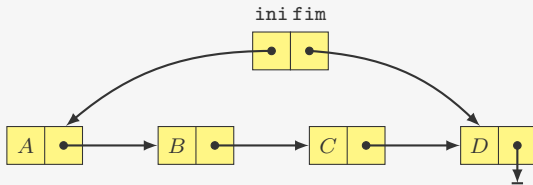
```
1 p_fila criar_fila() {
```

Fila: implementação com lista ligada



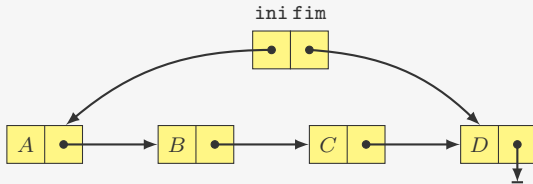
```
1 p_fila criar_fila() {  
2   p_fila f;
```

Fila: implementação com lista ligada



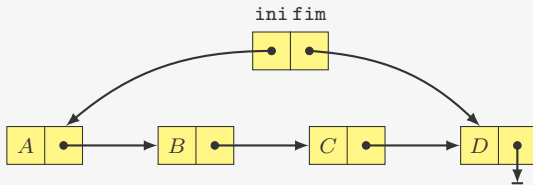
```
1 p_fila criar_fila() {  
2   p_fila f;  
3   f = malloc(sizeof(struct fila));
```

Fila: implementação com lista ligada



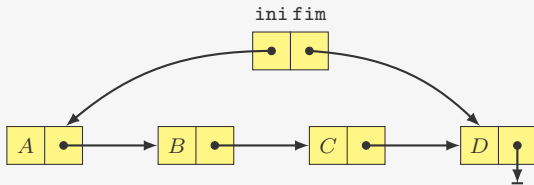
```
1 p_fila criar_fila() {  
2     p_fila f;  
3     f = malloc(sizeof(struct fila));  
4     f->ini = NULL;  
5     f->fim = NULL;
```

Fila: implementação com lista ligada



```
1 p_fila criar_fila() {  
2     p_fila f;  
3     f = malloc(sizeof(struct fila));  
4     f->ini = NULL;  
5     f->fim = NULL;  
6     return f;  
7 }
```

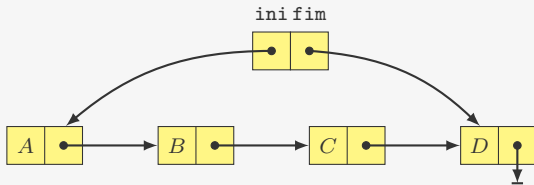

Fila: implementação com lista ligada



```
1 p_fila criar_fila() {  
2   p_fila f;  
3   f = malloc(sizeof(struct fila));  
4   f->ini = NULL;  
5   f->fim = NULL;  
6   return f;  
7 }
```

```
1 void destruir_fila(p_fila f) {
```

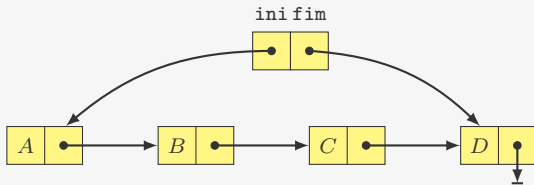
Fila: implementação com lista ligada



```
1 p_filha criar_filha() {
2   p_filha f;
3   f = malloc(sizeof(struct filha));
4   f->ini = NULL;
5   f->fim = NULL;
6   return f;
7 }
```

```
1 void destruir_filha(p_filha f) {
2   destruir_lista(f->ini);
```

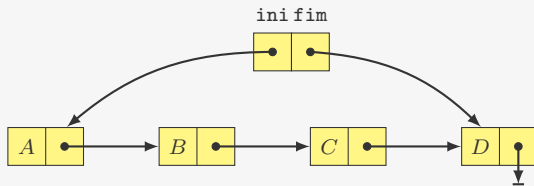
Fila: implementação com lista ligada



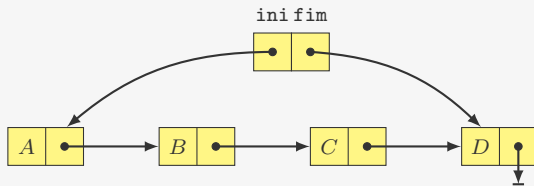
```
1 p_fila criar_fila() {
2   p_fila f;
3   f = malloc(sizeof(struct fila));
4   f->ini = NULL;
5   f->fim = NULL;
6   return f;
7 }
```

```
1 void destruir_fila(p_fila f) {
2   destruir_lista(f->ini);
3   free(f);
4 }
```

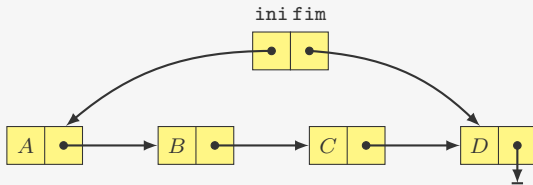
Fila: implementação com lista ligada



Fila: implementação com lista ligada



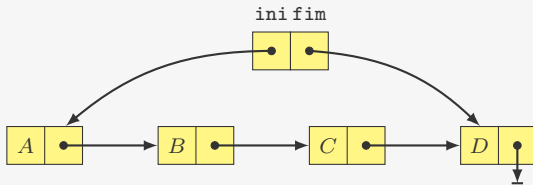
Fila: implementação com lista ligada



Inserir no final:

```
1 void enfileira(p_fila f, int x) {
```

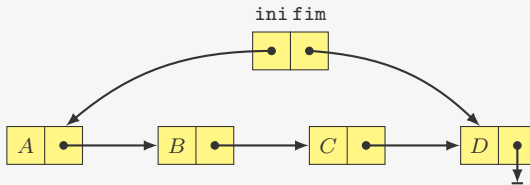
Fila: implementação com lista ligada



Inserir no final:

```
1 void enfileira(p_fila f, int x) {  
2     p_no novo;  
3     novo = malloc(sizeof(struct no));  
4     novo->dado = x;  
5     novo->prox = NULL;
```

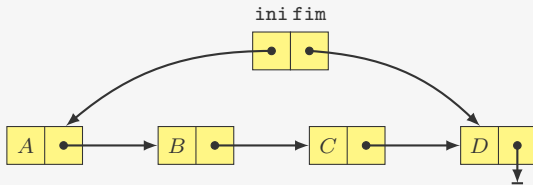
Fila: implementação com lista ligada



Inserir no final:

```
1 void enfileira(p_filha f, int x) {  
2     p_no novo;  
3     novo = malloc(sizeof(struct no));  
4     novo->dado = x;  
5     novo->prox = NULL;  
6     if (f->ini == NULL)  
7         f->ini = novo;
```

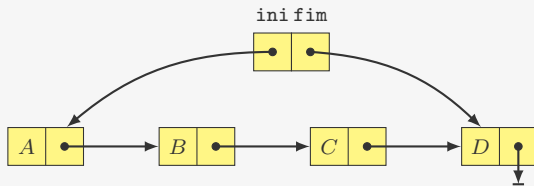

Fila: implementação com lista ligada



Inserir no final:

```
1 void enfileira(p_fila f, int x) {
2     p_no novo;
3     novo = malloc(sizeof(struct no));
4     novo->dado = x;
5     novo->prox = NULL;
6     if (f->ini == NULL)
7         f->ini = novo;
8     else
9         f->fim->prox = novo;
```

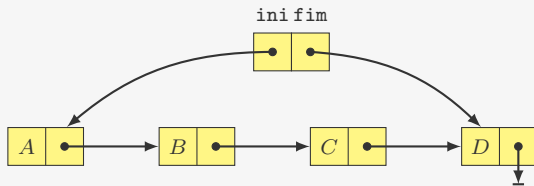
Fila: implementação com lista ligada



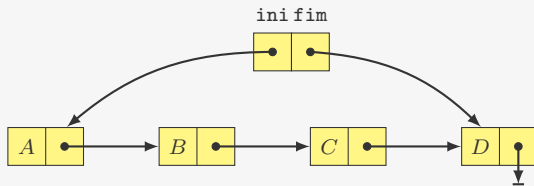
Inserir no final:

```
1 void enfileira(p_filha f, int x) {  
2     p_no novo;  
3     novo = malloc(sizeof(struct no));  
4     novo->dado = x;  
5     novo->prox = NULL;  
6     if (f->ini == NULL)  
7         f->ini = novo;  
8     else  
9         f->fim->prox = novo;  
10    f->fim = novo;  
11 }
```

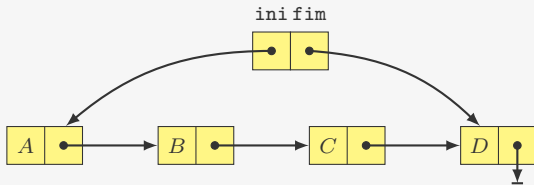
Fila: implementação com lista ligada



Fila: implementação com lista ligada



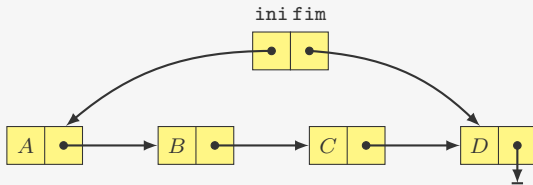
Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {
```

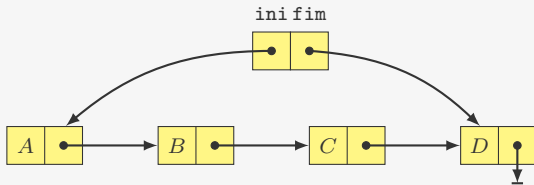
Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {  
2   p_no primeiro = f->ini;
```

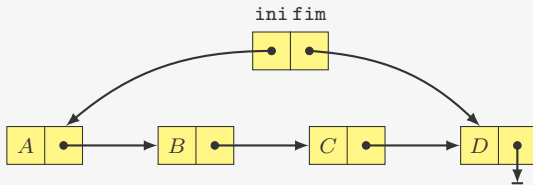
Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {  
2     p_no primeiro = f->ini;  
3     int x = primeiro->dado;
```

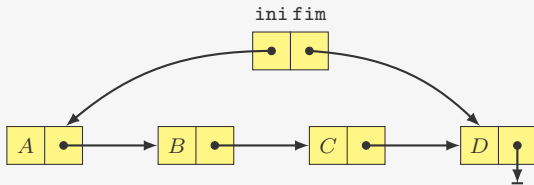
Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {  
2     p_no primeiro = f->ini;  
3     int x = primeiro->dado;  
4     f->ini = f->ini->prox;  
}
```

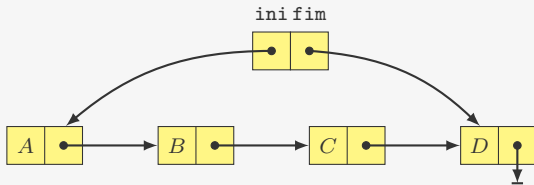

Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {  
2     p_no primeiro = f->ini;  
3     int x = primeiro->dado;  
4     f->ini = f->ini->prox;  
5     if (f->ini == NULL)  
6         f->fim = NULL;  
7     free(primeiro);  
}
```

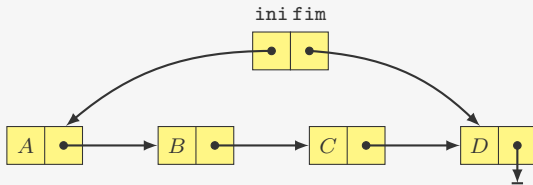
Fila: implementação com lista ligada



Remove do início:

```
1 int desenfileira(p_filha f) {
2     p_no primeiro = f->ini;
3     int x = primeiro->dado;
4     f->ini = f->ini->prox;
5     if (f->ini == NULL)
6         f->fim = NULL;
7     free(primeiro);
8     return x;
9 }
```

Fila: implementação com lista ligada

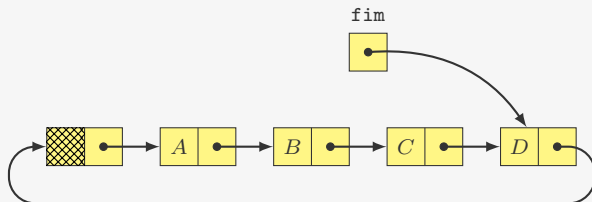


Remove do início:

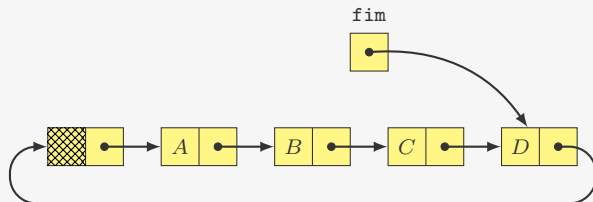
```
1 int desenfileira(p_filha f) {  
2     p_no primeiro = f->ini;  
3     int x = primeiro->dado;  
4     f->ini = f->ini->prox;  
5     if (f->ini == NULL)  
6         f->fim = NULL;  
7     free(primeiro);  
8     return x;  
9 }
```

Supõe que a fila não é vazia...

Fila: implementação com lista ligada (outra opção)

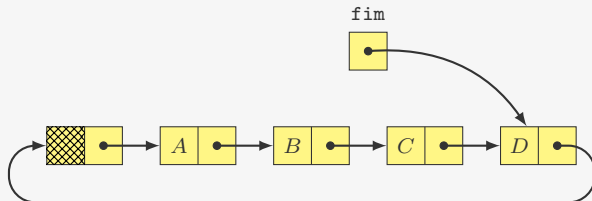


Fila: implementação com lista ligada (outra opção)



Enfileira:

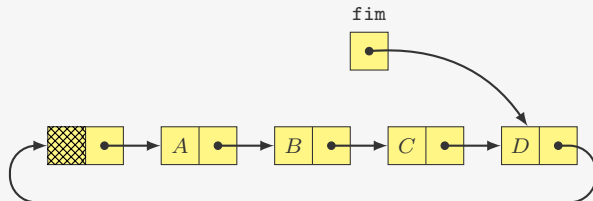
Fila: implementação com lista ligada (outra opção)



Enfileira:

- Fazer novo nó apontar para **fim**->**prox**

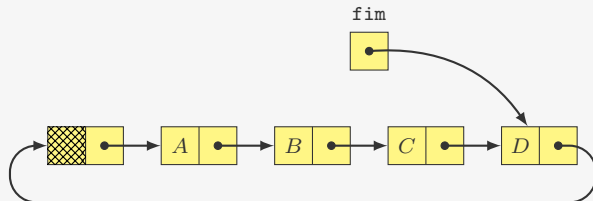
Fila: implementação com lista ligada (outra opção)



Enfileira:

- Fazer novo nó apontar para `fim->prox`
- Atualizar campo `fim->prox` para novo nó

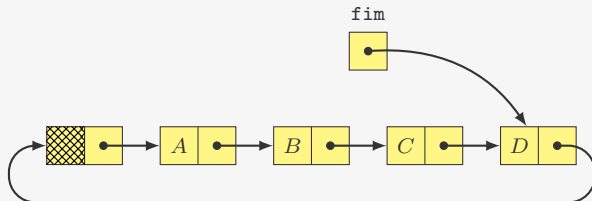
Fila: implementação com lista ligada (outra opção)



Enfileira:

- Fazer novo nó apontar para `fim->prox`
- Atualizar campo `fim->prox` para novo nó
- Mudar `fim` para apontar para o novo nó

Fila: implementação com lista ligada (outra opção)

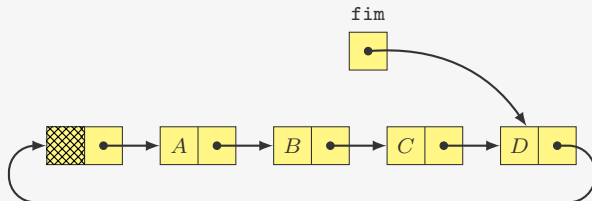


Enfileira:

- Fazer novo nó apontar para `fim->prox`
- Atualizar campo `fim->prox` para novo nó
- Mudar `fim` para apontar para o novo nó

Desenfileira:

Fila: implementação com lista ligada (outra opção)



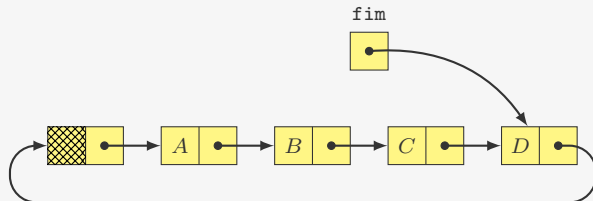
Enfileira:

- Fazer novo nó apontar para `fim->prox`
- Atualizar campo `fim->prox` para novo nó
- Mudar `fim` para apontar para o novo nó

Desenfileira:

- Basta remover o nó seguinte ao nó dummy

Fila: implementação com lista ligada (outra opção)



Enfileira:

- Fazer novo nó apontar para `fim->prox`
- Atualizar campo `fim->prox` para novo nó
- Mudar `fim` para apontar para o novo nó

Desenfileira:

- Basta remover o nó seguinte ao nó dummy
 - i.e., `fim->prox->prox`

Fila: implementação com vetor

Primeira ideia:

Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$

Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável `ini` indica o começo da fila

Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável `ini` indica o começo da fila
- Variável `fim` indica o fim da fila

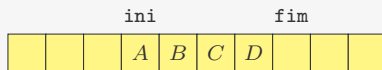
Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável `ini` indica o começo da fila
- Variável `fim` indica o fim da fila



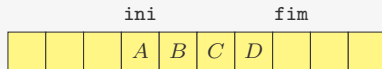
Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável `ini` indica o começo da fila
- Variável `fim` indica o fim da fila



E se, ao inserir, tivermos espaço apenas à esquerda de `ini`?

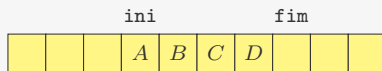
Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável `ini` indica o começo da fila
- Variável `fim` indica o fim da fila



E se, ao inserir, tivermos espaço apenas à esquerda de `ini`?

- podemos mover toda a fila para o começo do vetor

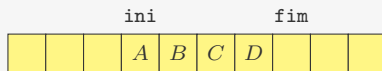
Fila: implementação com vetor

Primeira ideia:

- Inserimos no final do vetor: $O(1)$
- Removemos do começo do vetor: $O(n)$

Segunda ideia:

- Variável **ini** indica o começo da fila
- Variável **fim** indica o fim da fila



E se, ao inserir, tivermos espaço apenas à esquerda de **ini**?

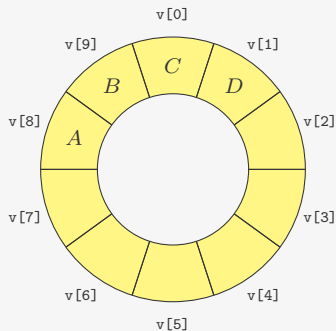
- podemos mover toda a fila para o começo do vetor
- mas isso leva tempo $O(n)$...

Fila: implementação com vetor (fila circular)

Solução: considerar o vetor de tamanho N de maneira **circular**

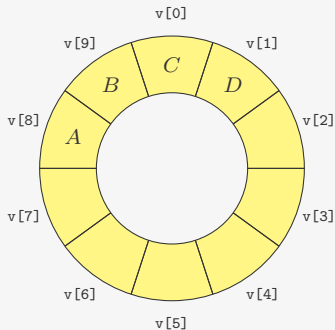
Fila: implementação com vetor (fila circular)

Solução: considerar o vetor de tamanho **N** de maneira **circular**



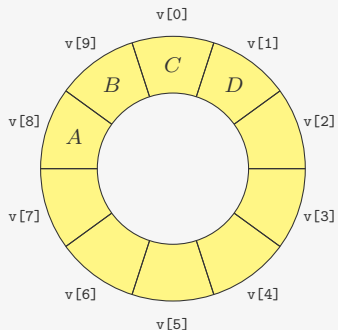
Fila: implementação com vetor (fila circular)

Solução: considerar o vetor de tamanho N de maneira **circular**



As manipulações de índices são realizadas módulo N

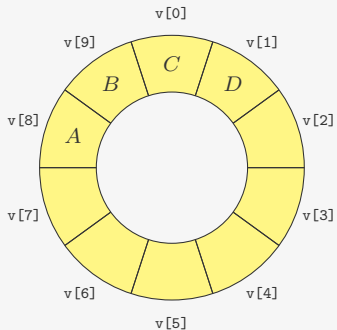
Fila circular — Estrutura



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     int *v;  
5     int ini, fim, N, tamanho;  
6 };
```

Fila circular — Estrutura

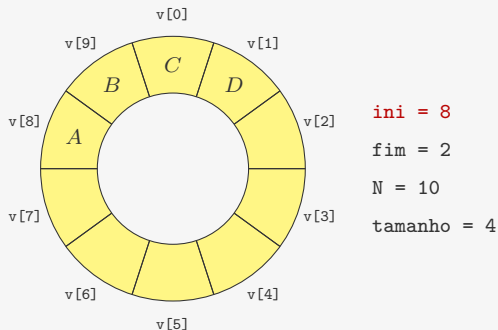


ini = 8
fim = 2
N = 10
tamanho = 4

```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     int *v;  
5     int ini, fim, N, tamanho;  
6 };
```

vetor para armazenar os dados

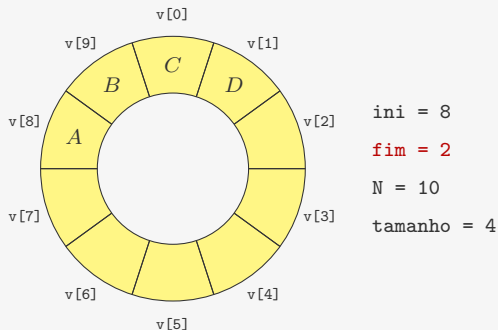
Fila circular — Estrutura



```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     int *v;  
5     int ini, fim, N, tamanho;  
6 };
```

início da fila (posição da próxima remoção)

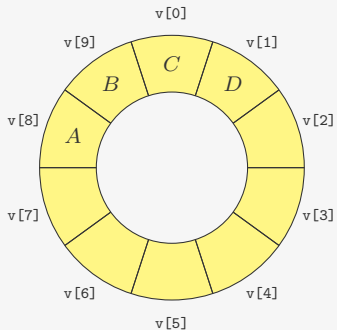
Fila circular — Estrutura



```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     int *v;  
5     int ini, fim, N, tamanho;  
6 };
```

← fim da fila (posição da próxima inserção)

Fila circular — Estrutura



ini = 8

fim = 2

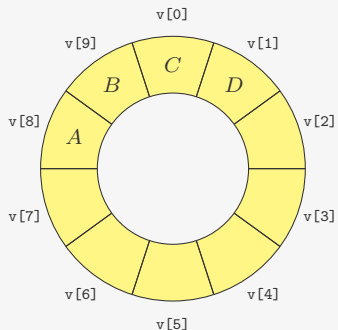
N = 10

tamanho = 4

```
1 typedef struct fila *p_fila;
2
3 struct fila {
4     int *v;
5     int ini, fim, N, tamanho;
6 };
```

tamanho do vetor alocado

Fila circular — Estrutura



ini = 8

fim = 2

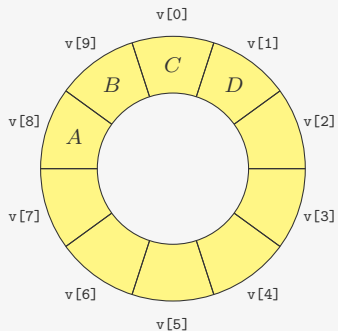
N = 10

tamanho = 4

```
1 typedef struct fila *p_fila;  
2  
3 struct fila {  
4     int *v;  
5     int ini, fim, N, tamanho;  
6 };
```

tamanho da fila (número de elementos)

Fila circular — Criando



`ini = 8`

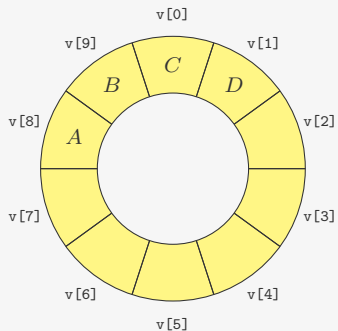
`fim = 2`

`N = 10`

`tamanho = 4`

```
1 p_fila criar_fila(int N) {
```

Fila circular — Criando



`ini = 8`

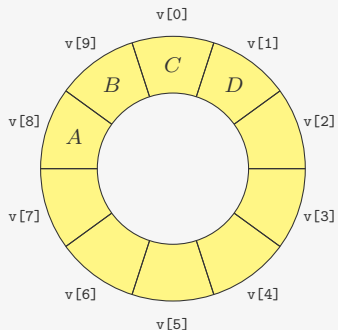
`fim = 2`

`N = 10`

`tamanho = 4`

```
1 p_fila criar_fila(int N) {  
2   p_fila f;
```

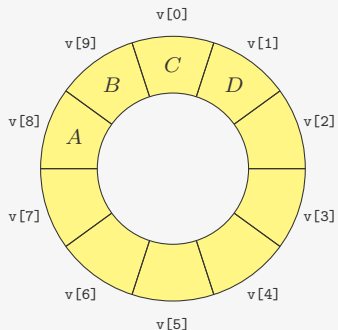

Fila circular — Criando



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 p_fila criar_fila(int N) {  
2   p_fila f;  
3   f = malloc(sizeof(struct fila));
```

Fila circular — Criando



ini = 8

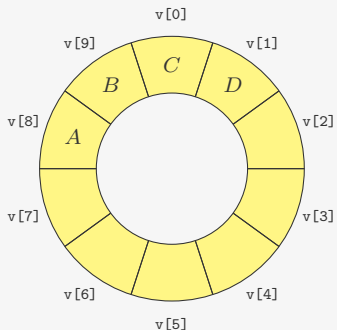
fim = 2

N = 10

tamanho = 4

```
1 p_fila criar_fila(int N) {  
2   p_fila f;  
3   f = malloc(sizeof(struct fila));  
4   f->v = malloc(N * sizeof(int));
```

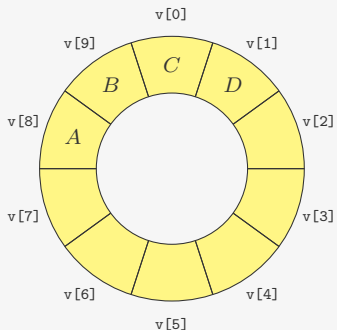
Fila circular — Criando



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 p_fila criar_fila(int N) {  
2     p_fila f;  
3     f = malloc(sizeof(struct fila));  
4     f->v = malloc(N * sizeof(int));  
5     f->ini = 0;  
6     f->fim = 0;  
7     f->N = N;  
8     f->tamanho = 0;
```

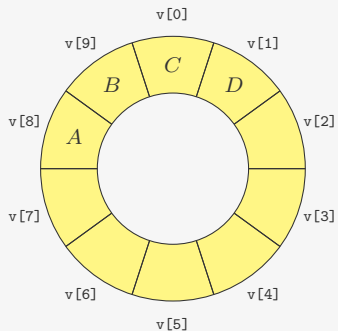
Fila circular — Criando



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 p_fila criar_fila(int N) {
2   p_fila f;
3   f = malloc(sizeof(struct fila));
4   f->v = malloc(N * sizeof(int));
5   f->ini = 0;
6   f->fim = 0;
7   f->N = N;
8   f->tamanho = 0;
9   return f;
10 }
```

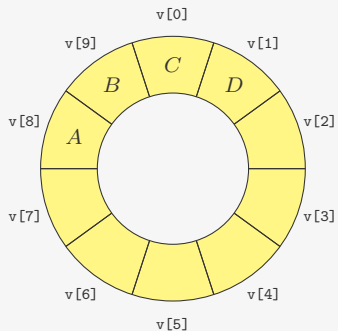
Fila circular — Enfileira



`ini = 8`
`fim = 2`
`N = 10`
`tamanho = 4`

```
1 void enfileira(p_fila f, int x) {
```

Fila circular — Enfileira



`ini = 8`

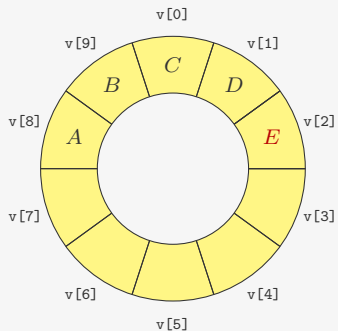
`fim = 2`

`N = 10`

`tamanho = 4`

```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;
```

Fila circular — Enfileira



ini = 8

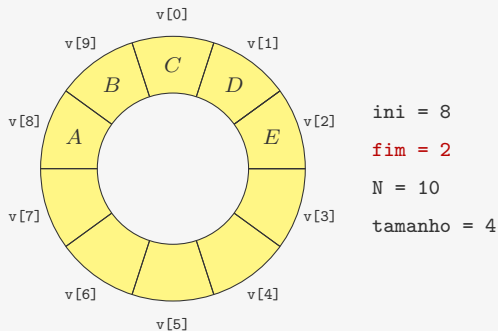
fim = 2

N = 10

tamanho = 4

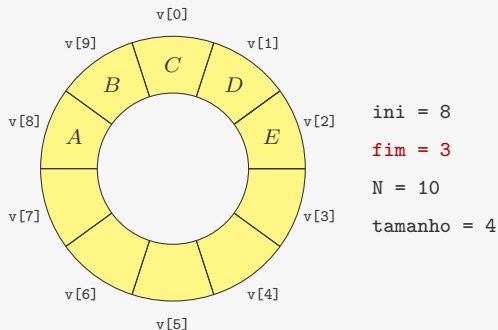
```
1 void enfileira(p_fila f, int x) {  
2   f->v[f->fim] = x;
```

Fila circular — Enfileira



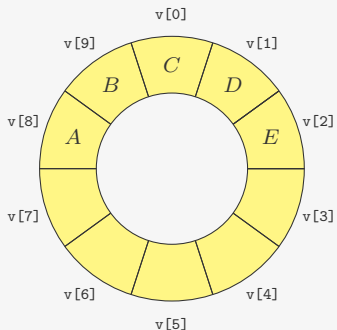
```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;  
3     f->fim = (f->fim + 1) % f->N;  
}
```


Fila circular — Enfileira



```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;  
3     f->fim = (f->fim + 1) % f->N;  
}
```

Fila circular — Enfileira



`ini = 8`

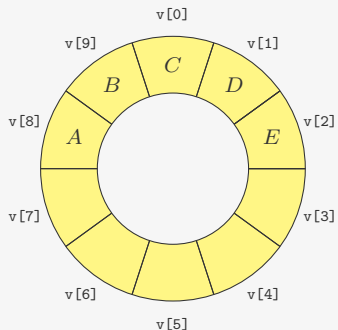
`fim = 3`

`N = 10`

`tamanho = 4`

```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;  
3     f->fim = (f->fim + 1) % f->N;  
4     f->tamanho++;  
5 }
```

Fila circular — Enfileira



ini = 8

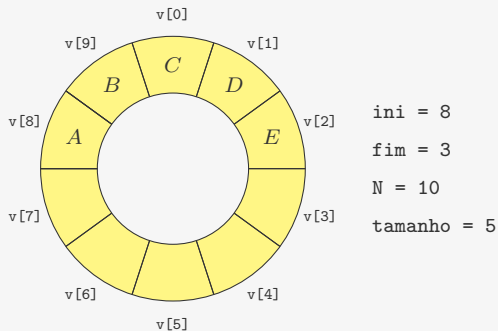
fim = 3

N = 10

tamanho = 5

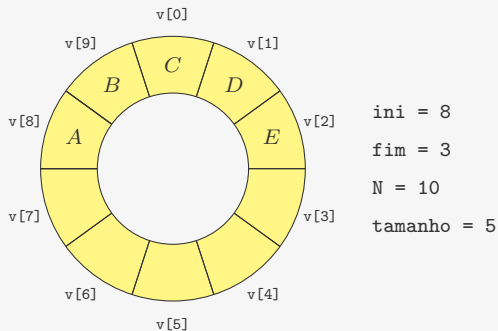
```
1 void enfileira(p_fila f, int x) {  
2     f->v[f->fim] = x;  
3     f->fim = (f->fim + 1) % f->N;  
4     f->tamanho++;  
5 }
```

Fila circular — Desenfileira



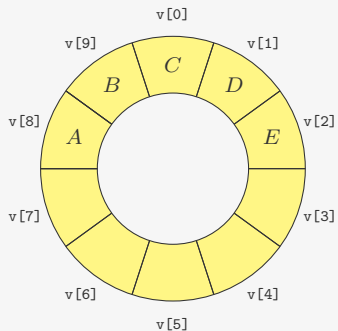
```
1 int desenfileira(p_fila f) {
```

Fila circular — Desenfileira



```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];
```

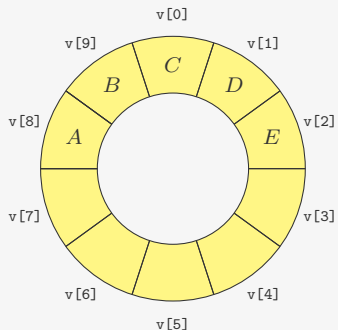
Fila circular — Desenfileira



`ini = 8`
`fim = 3`
`N = 10`
`tamanho = 5`
`x = A`

```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];
```

Fila circular — Desenfileira



`ini = 8`

`fim = 3`

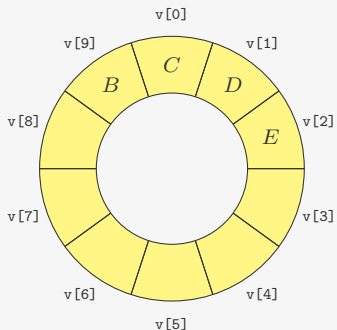
`N = 10`

`tamanho = 5`

`x = A`

```
1 int desenfileira(p_filha f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;
```

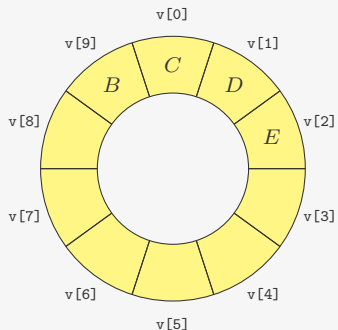
Fila circular — Desenfileira



`ini = 9`
`fim = 3`
`N = 10`
`tamanho = 5`
`x = A`

```
1 int desenfileira(p_filha f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;  
}
```


Fila circular — Desenfileira



`ini = 9`

`fim = 3`

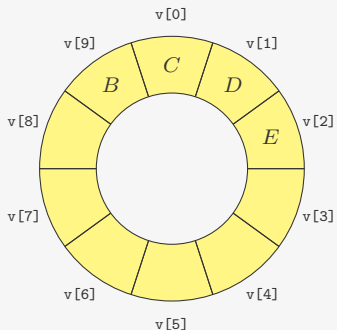
`N = 10`

`tamanho = 5`

`x = A`

```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;  
4     f->tamanho--;
```

Fila circular — Desenfileira



`ini = 9`

`fim = 3`

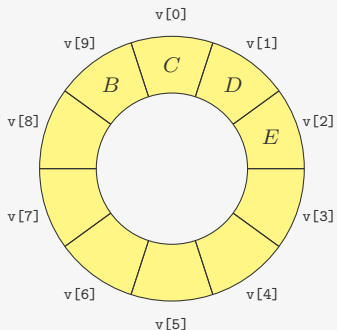
`N = 10`

`tamanho = 4`

`x = A`

```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;  
4     f->tamanho--;
```

Fila circular — Desenfileira



`ini = 9`
`fim = 3`
`N = 10`
`tamanho = 4`
`x = A`

```
1 int desenfileira(p_fila f) {  
2     int x = f->v[f->ini];  
3     f->ini = (f->ini + 1) % f->N;  
4     f->tamanho--;  
5     return x;  
6 }
```

Um cliente simples

Um cliente simples

```
1 int main() {  
2     int n, x, i;  
3     p_fila f;
```

Um cliente simples

```
1 int main() {  
2     int n, x, i;  
3     p_fila f;  
4     f = criar_fila(100);
```

Um cliente simples

```
1 int main() {  
2     int n, x, i;  
3     p_fila f;  
4     f = criar_fila(100);  
5     scanf("%d", &n);
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
```


Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = desenfileira(f);
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = desenfileira(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = desenfileira(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Qual é o problema do código acima?

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enqueue(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = dequeue(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Qual é o problema do código acima?

- E se **n** for maior do que **100**?

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enqueue(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = dequeue(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Qual é o problema do código acima?

- E se **n** for maior do que **100**?
 - poderíamos usar listas ligadas

Um cliente simples

```
1 int main() {
2     int n, x, i;
3     p_fila f;
4     f = criar_fila(100);
5     scanf("%d", &n);
6     for (i = 0; i < n; i++) {
7         scanf("%d", &x);
8         enfileira(f, x);
9     }
10    while(!fila_vazia(f)) {
11        x = desenfileira(f);
12        printf("%d ", x);
13    }
14    printf("\n");
15    destroi_fila(f);
16    return 0;
17 }
```

Qual é o problema do código acima?

- E se **n** for maior do que **100**?
 - poderíamos usar listas ligadas
 - ou um vetor dinâmico (com alguns cuidados)

Exemplos de aplicações

Algumas aplicações de filas:

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores

Exemplos de aplicações

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores
- Percurso de estruturas de dados complexas (grafos, etc.)

Pilha

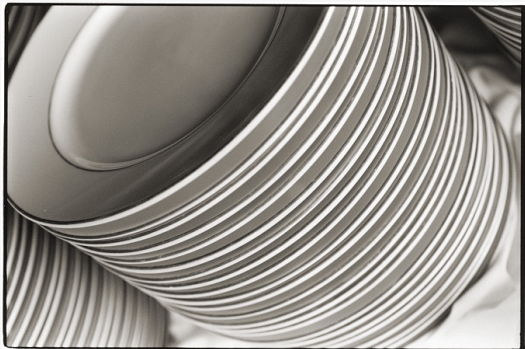
- Remove primeiro objetos **inseridos há menos tempo**

Pilha

- Remove primeiro objetos **inseridos há menos tempo**
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair

Pilha

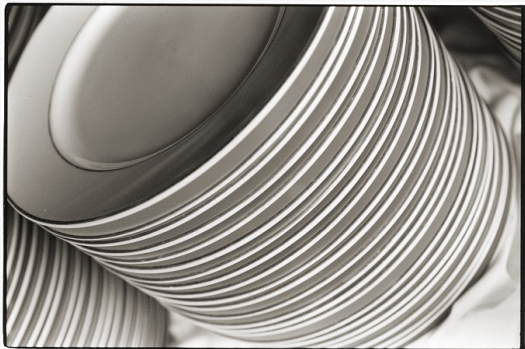
- Remove primeiro objetos **inseridos há menos tempo**
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair



É como uma pilha de pratos:

Pilha

- Remove primeiro objetos **inseridos há menos tempo**
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair

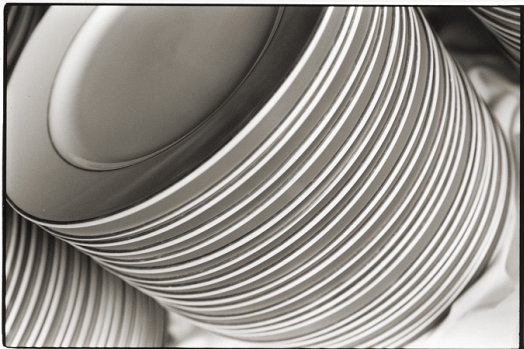


É como uma pilha de pratos:

- **Empilha** os pratos limpos sobre os que já estão na pilha

Pilha

- Remove primeiro objetos **inseridos há menos tempo**
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair



É como uma pilha de pratos:

- **Empilha** os pratos limpos sobre os que já estão na pilha
- **Desempilha** o prato de cima para usar

Pilha

Operações:

Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha

Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo:



Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha(A)**

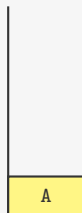


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha**(A)

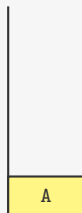


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha**(B)

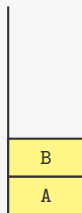


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha**(B)

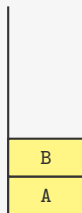


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Desempilha()**

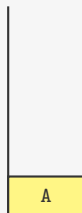


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Desempilha()**



Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha**(C)

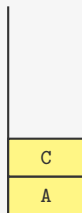


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha(C)**

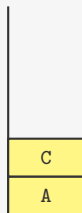


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha(D)**

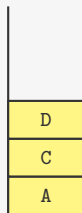


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Empilha(D)**

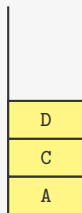


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Desempilha()**

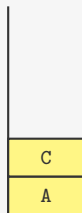


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Desempilha()**

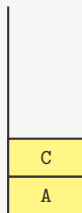


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

Exemplo: **Desempilha()**

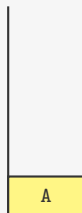


Pilha

Operações:

- **Empilha** (*push*): adiciona no topo da pilha
- **Desempilha** (*pop*): remove do topo da pilha

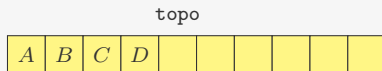
Exemplo: **Desempilha()**



Pilha: implementação com vetor

Definição:

```
1 typedef struct pilha *p_pilha;  
2  
3 struct pilha {  
4     int *v;  
5     int topo;  
6 };
```



Pilha: implementação com vetor

Definição:

```
1 typedef struct pilha *p_pilha;  
2  
3 struct pilha {  
4     int *v;  
5     int topo;  
6 };
```

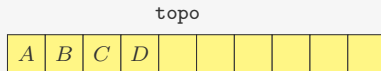


vetor para armazenar os dados

Pilha: implementação com vetor

Definição:

```
1 typedef struct pilha *p_pilha;  
2  
3 struct pilha {  
4     int *v;  
5     int topo;  
6 };
```



fim da pilha (posição da próxima inserção)

Pilha: implementação com vetor

Definição:

```
1 typedef struct pilha *p_pilha;
2
3 struct pilha {
4     int *v;
5     int topo;
6 };
```



Inserção:

```
1 void empilhar(p_pilha p, int i) {
2     p->v[p->topo] = i;
3     p->topo++;
4 }
```

Pilha: implementação com vetor

Definição:

```
1 typedef struct pilha *p_pilha;
2
3 struct pilha {
4     int *v;
5     int topo;
6 };
```



Inserção:

```
1 void empilhar(p_pilha p, int i) {
2     p->v[p->topo] = i;
3     p->topo++;
4 }
```

Remoção:

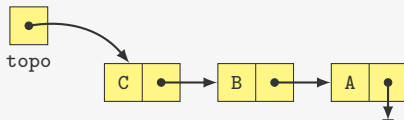
```
1 int desempilhar(p_pilha p) {
2     p->topo--;
3     return p->v[p->topo];
4 }
```

Pilha: implementação com lista ligada

Após empilhar A, B e C:

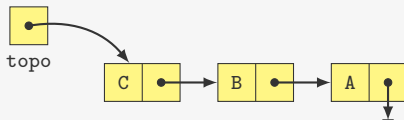
Pilha: implementação com lista ligada

Após empilhar A, B e C:



Pilha: implementação com lista ligada

Após empilhar A, B e C:

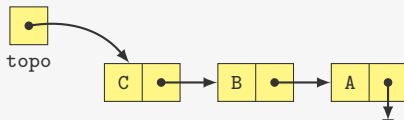


Estrutura:

```
1 typedef struct pilha *p_pilha;  
2  
3 struct pilha {  
4     p_no topo;  
5 };
```

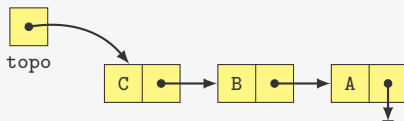
Pilha: implementação com lista ligada

Após empilhar A, B e C:



Pilha: implementação com lista ligada

Após empilhar A, B e C:

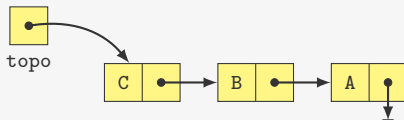


Empilhando:

```
1 void empilhar(p_pilha pilha, int x) {  
2     p_no novo = malloc(sizeof(struct no));  
3     novo->dado = x;  
4     novo->prox = pilha->topo;  
5     pilha->topo = novo;  
6 }
```

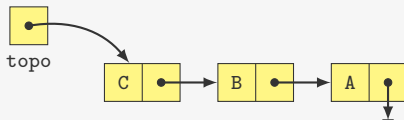
Pilha: implementação com lista ligada

Após empilhar A, B e C:



Pilha: implementação com lista ligada

Após empilhar A, B e C:



Desempilhando:

```
1 int desempilhar(p_pilha pilha) {  
2     p_no topo = pilha->topo;  
3     int x = topo->dado;  
4     pilha->topo = pilha->topo->prox;  
5     free(topo);  
6     return x;  
7 }
```

Exemplos de aplicações

Algumas aplicações de pilhas:

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (grafos, etc.)

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (grafos, etc.)
- Recursão

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (grafos, etc.)
- Recursão

Veremos algumas dessas aplicações na próxima unidade

Exercício

Implemente fila como uma lista ligada circular com cabeça.

Solução

```
1 typedef struct no {
2     int dado;
3     struct no *prox;
4 } No;
5
6 typedef No * p_no;
7
8 p_no cria_fila() {
9     p_no novo = malloc(sizeof(No));
10    novo->prox = novo;
11    return novo;
12 }
13
14 p_no enfilera(p_no fim, int dado) {
15     p_no novo = malloc(sizeof(No));
16     novo->dado = dado;
17     novo->prox = fim->prox;
18     fim->prox = novo;
19     return novo;
20 }
21
22 int desenfilera(p_no fim) {
23     p_no aux = fim->prox->prox;
24     int dado = aux->dado;
25     fim->prox->prox = aux->prox;
26     free(aux);
27     return dado;
28 }
```

Exercício

Um *deque* (*double-ended queue*) é uma estrutura de dados com as operações: `insere_inicio`, `insere_fim`, `remove_inicio`, `remove_fim`.

Implemente um *deque* utilizando listas ligadas.

Solução

```
1  typedef struct no {
2      int dado;
3      struct no *prox, *ant;
4  } No;
5
6  typedef No * p_no;
7
8  typedef struct {
9      p_no inicio, fim;
10 } Deque;
11
12 typedef Deque * p_deque;
13
14 p_deque cria_deque() {
15     p_deque d = malloc(sizeof(Deque));
16     d->inicio = d->fim = NULL;
17     return d;
18 }
19
20 void insere_inicio(p_deque d, int dado) {
21     p_no novo = malloc(sizeof(No));
22     novo->dado = dado;
23     novo->prox = d->inicio;
24     novo->ant = NULL;
25     if (d->inicio != NULL)
26         d->inicio->ant = novo;
27     else
28         d->fim = novo;
29     d->inicio = novo;
30 }
```

Solução

```
31 void insere_fim(p_deque d, int dado) {
32     p_no novo = malloc(sizeof(No));
33     novo->dado = dado;
34     novo->prox = NULL;
35     novo->ant = d->fim;
36     if (d->fim != NULL)
37         d->fim->prox = novo;
38     else
39         d->inicio = novo;
40     d->fim = novo;
41 }
42
43 int remove_inicio(p_deque d) {
44     p_no aux = d->inicio;
45     int dado = aux->dado;
46     d->inicio = d->inicio->prox;
47     if (d->inicio != NULL)
48         d->inicio->ant = NULL;
49     else
50         d->fim = NULL;
51     free(aux);
52     return dado;
53 }
54
55 int remove_fim(p_deque d) {
56     p_no aux = d->fim;
57     int dado = aux->dado;
58     d->fim = d->fim->ant;
59     if (d->fim != NULL)
60         d->fim->prox = NULL;
61     else
62         d->inicio = NULL;
63     free(aux);
64     return dado;
65 }
```

Dúvidas?