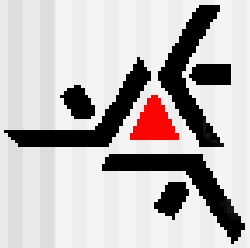


**Universidade Estadual de Maringá**
**Centro de Tecnologia**
**Departamento de Informática**

# Filas e Pilhas

Prof. Flávio Rogério Uber

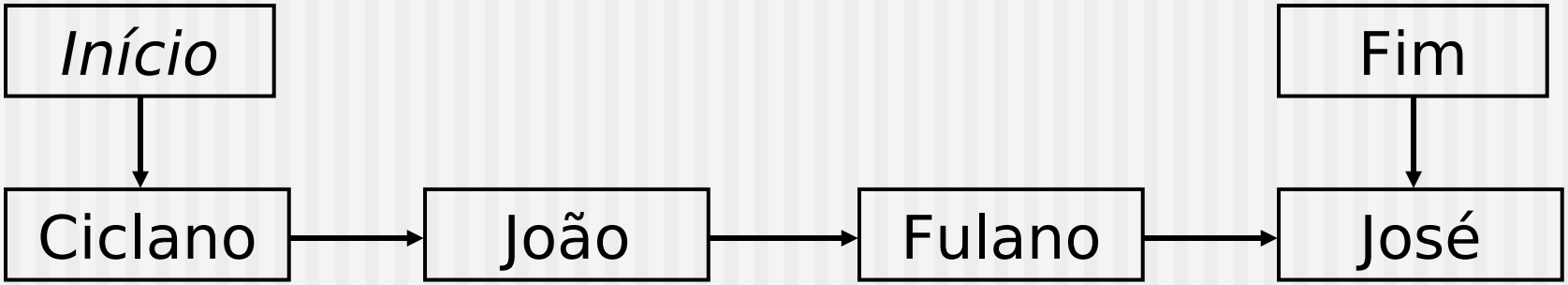
Material: Prof. Yandre Maldonado e Gomes da Costa

# Filas

- Filas são listas com disciplina de acesso;
- Todo elemento que entra numa fila entra no fim da mesma, e todo elemento que sai, sai do início da mesma;
- Também conhecida listas FIFO (*first-in-first-out*);
- Principal finalidade: registrar a ordem de chegada dos seus elementos;
  - Aplicações:
    - Escalonamento de processos em sistemas operacionais;
    - Fila de impressão.

# Filas

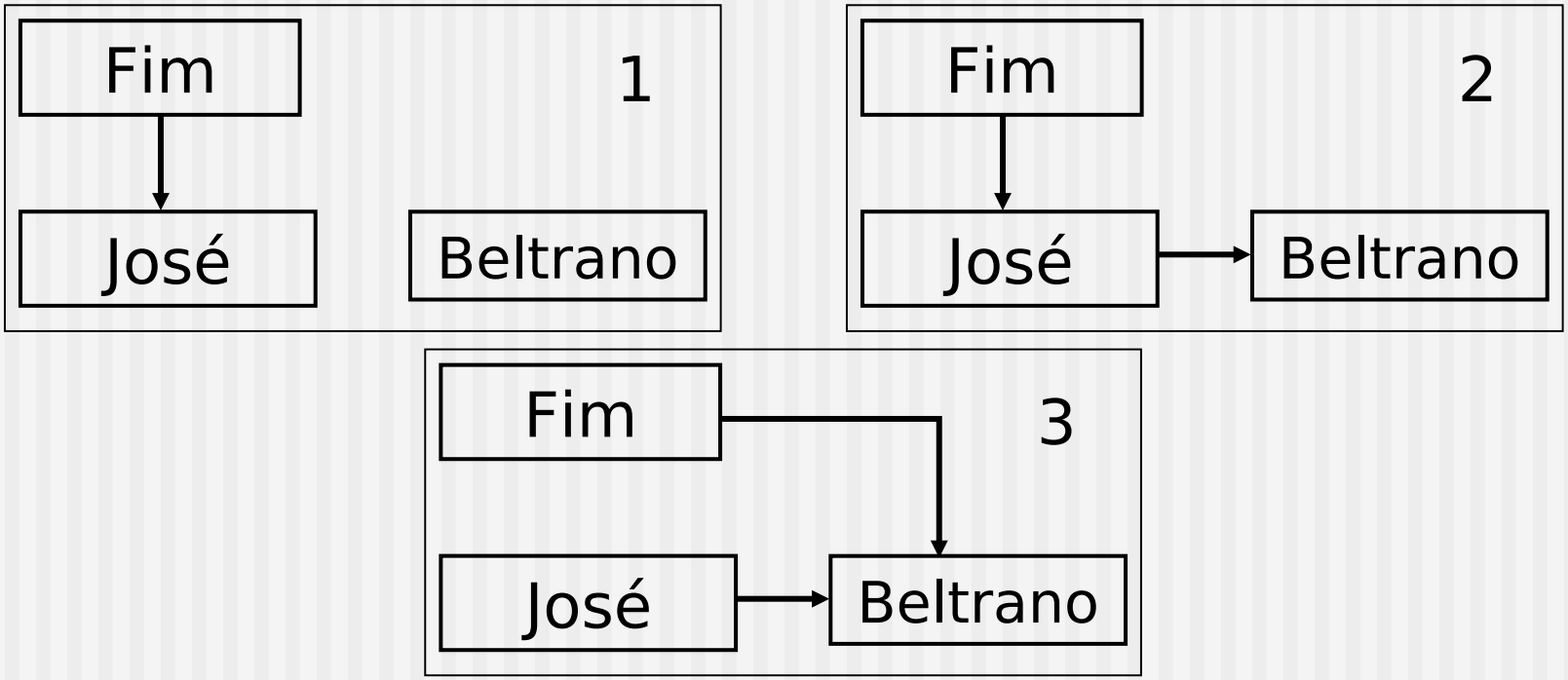
- Considere a seguinte fila:




- Observe que não há ordenação por chave.

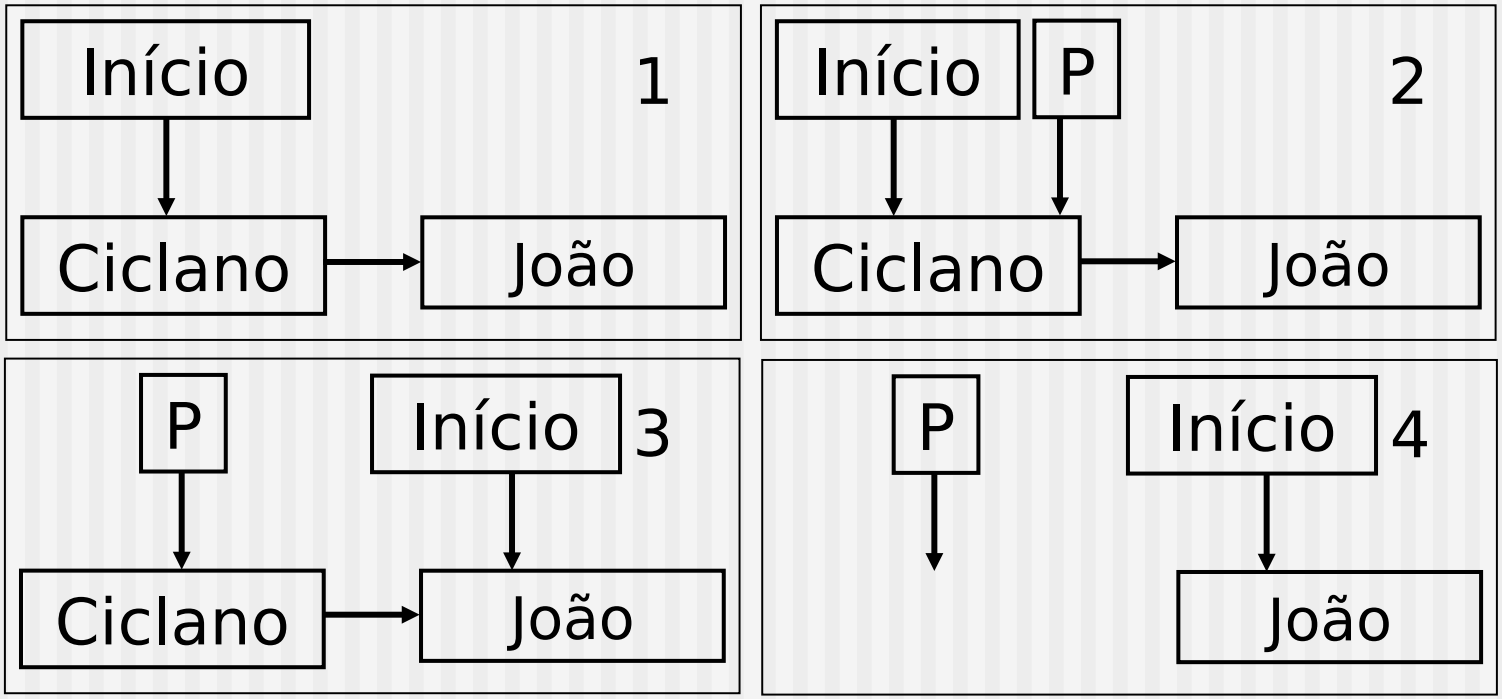
# Filas

- Inserção
  - Feita com o auxílio de uma variável (ponteiro) que indica a posição do último elemento da fila:

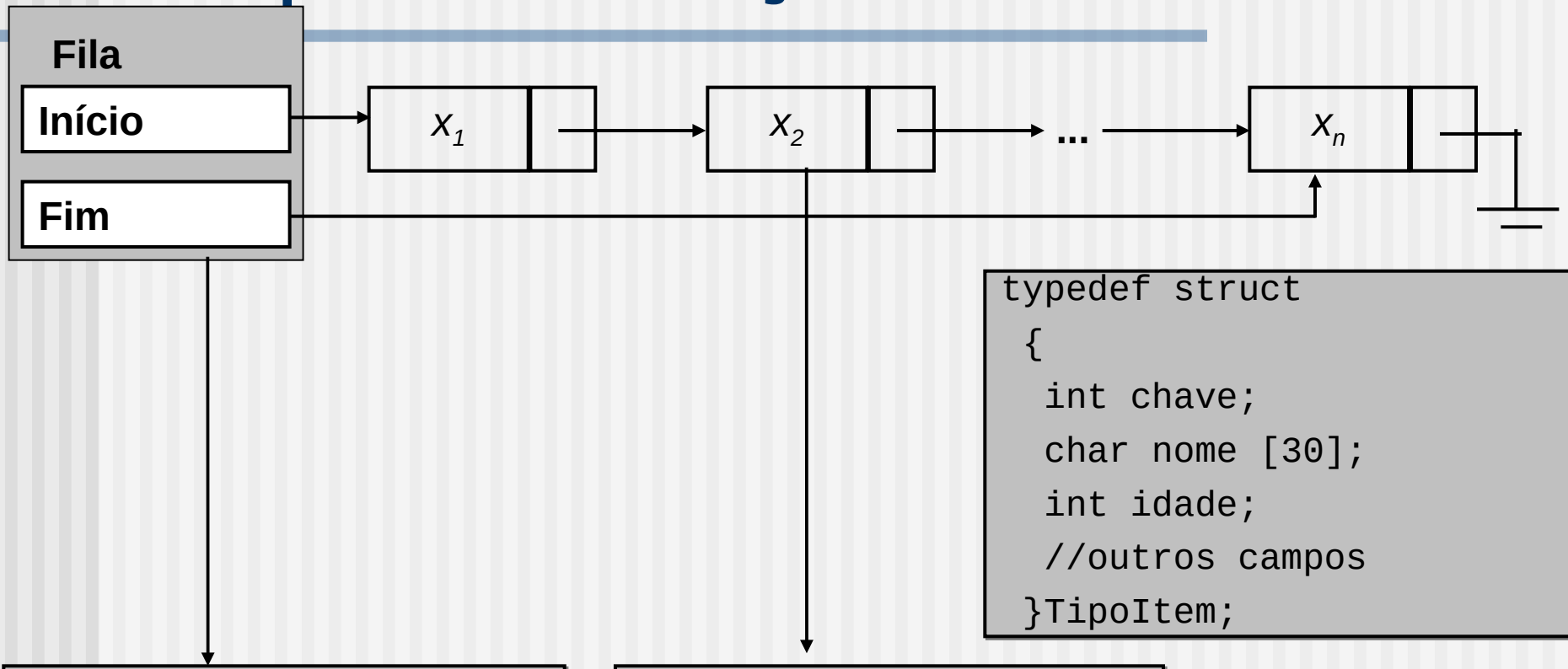


# Filas

- Remoção
  - Feita no início da fila, com o auxílio de uma variável (ponteiro) que indica a posição do primeiro elemento da fila:



# Implementação




```
typedef struct
 {
  int chave;
  char nome [30];
  int idade;
  //outros campos
 }TipoItem;
```

```
typedef struct
 {
  Celula* Inicio;
  Celula* Fim;
 }TipoFila;
```

```
struct Celula
 {
  TipoItem Item;
  Celula* Prox;
 };
```

# Implementação

```
void FFVazia(TipoFila *Fila)
{
	Fila->Inicio = NULL;
	Fila->Fim = Fila->Inicio;
}
```

```
int Vazia(TipoFila Fila)
{
	return (Fila.Inicio == NULL);
}
```

```
//Com célula cabeça
void FFVazia(TipoFila *Fila)
{
	Fila->Inicio = (struct Celula*)
malloc(sizeof(Celula));
	Fila->Fim = Fila->Inicio;
	Fila->Inicio->Prox = NULL;
}
```

```
//Com célula cabeça
int Vazia(TipoFila Fila)
{
	return (Fila.Inicio == Fila.Fim);
}
```

# Implementação

```
void Enfileira(TipoItem x, TipoFila *Fila)
{
      if (Fila->Fim == NULL)
            Fila->Fim = (struct Celula*) malloc(sizeof(Celula));
      else
            Fila->Fim->Prox = (struct Celula*) malloc(sizeof(Celula));
      Fila->Fim = Fila->Fim->Prox;
      Fila->Fim->Item = x;
      Fila->Fim->Prox = NULL;
      if(Fila->Inicio==NULL)
              Fila->Inicio=Fila->Fim;
}
```

# Implementação

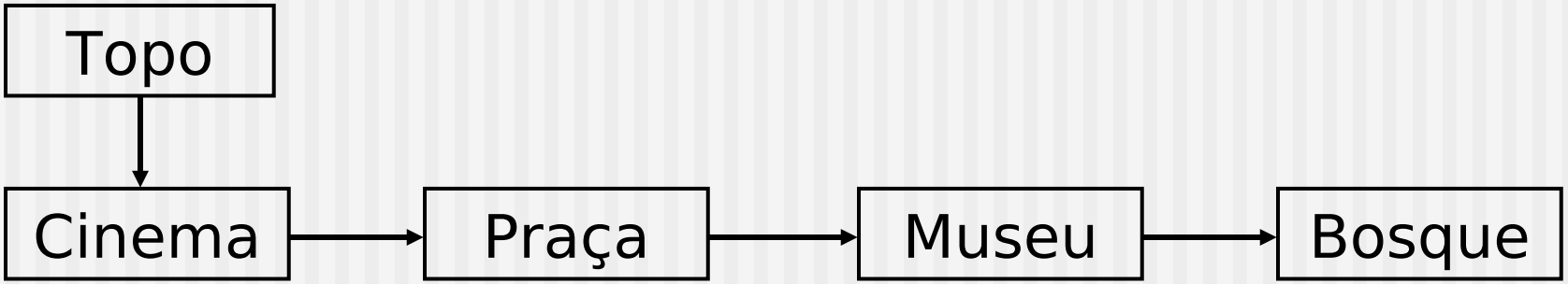
```
void Desenfileira(TipoFila *Fila, TipoItem *Item)
{
     struct Celula *q;
     if (Vazia(*Fila))
     {
          printf(" Erro:   fila está vazia\n");
          return;
     }
    q = Fila->Inicio;
    Fila->Inicio = Fila->Inicio->Prox;
    *Item = Fila->Inicio->Item;
    free(q);
    if(Fila->Inicio==NULL)
           Fila->Fim = NULL;
}
```

# Pilhas

- Pilhas são listas com disciplina de acesso;
- Todo elemento que entra numa pilha entra no fim da mesma, e todo elemento que sai, também sai do final da mesma;
- Um elemento só sai quando todos aqueles que entraram depois dele tiverem sido removidos;
- Também conhecida como listas LIFO (*last-in-first-out*);
- Principal finalidade: tornar disponíveis primeiro os elementos mais recentes;
  - Aplicações:
    - Avançar e Voltar em navegadores;
    - Desfazer e refazer em editores.

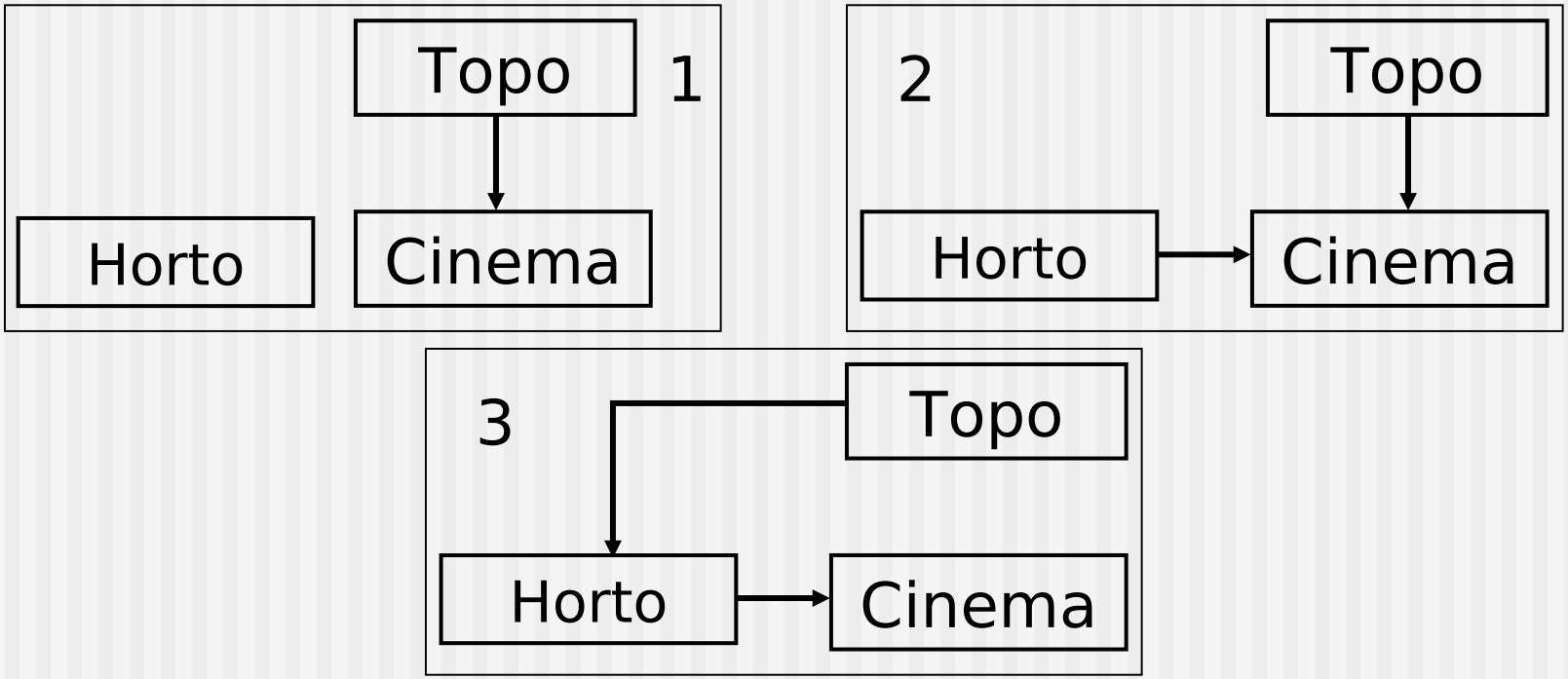
# Pilhas

- As operações (inserção/remoção) são feitas com base numa variável que indica o topo da pilha.



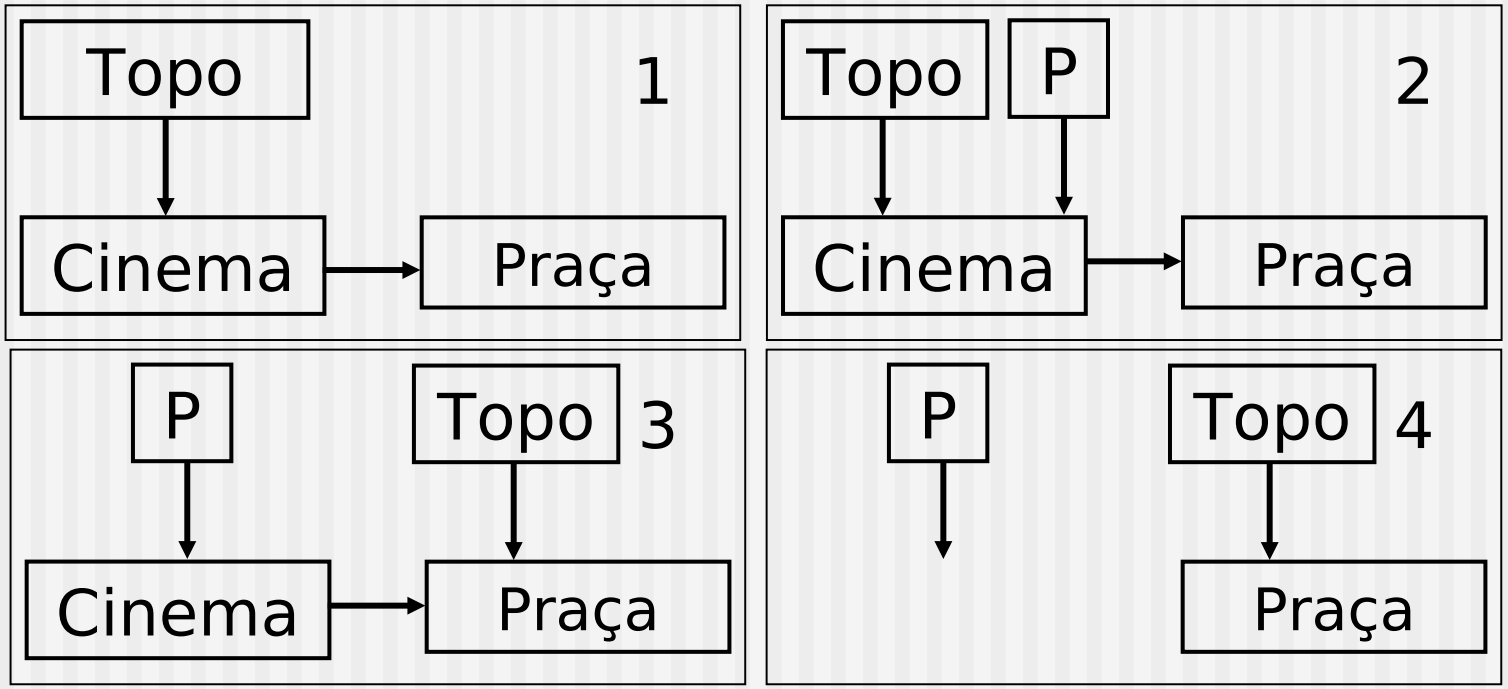
# Pilhas

- Inserção
  - As inserções, também denominadas empilhamentos, são realizadas no final com o auxílio da variável topo:

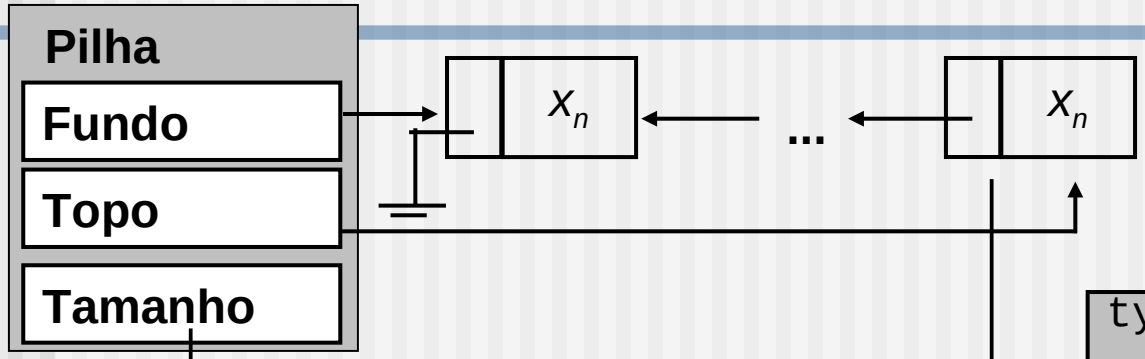


# Pilhas

- Remoção
  - As remoções, também denominadas desempilhamentos, são realizadas no final (topo) da pilha:



# Implementação




```
typedef struct {
        Celula *Fundo;
        Celula *Topo;
        int Tamanho;
} TipoPilha;
```

```
typedef struct
 {
  int chave;
  char nome [30];
  int idade;
  //outros campos
 }TipoItem;
```

```
struct Celula
 {
  TipoItem Item;
  struct Celula *Prox;
 };
```

# Implementação

```
void FPVazia(TipoPilha *Pilha)
{
    Pilha->Topo = NULL
    Pilha->Fundo = Pilha->Topo;
    Pilha->Tamanho = 0;
}
```

```
//com célula cabeça
void FPVazia(TipoPilha *Pilha)
{
    Pilha->Topo = (struct Celula*) malloc(sizeof(Celula));
    Pilha->Fundo = Pilha->Topo;
    Pilha->Topo->Prox = NULL;
    Pilha->Tamanho = 0;
}
```

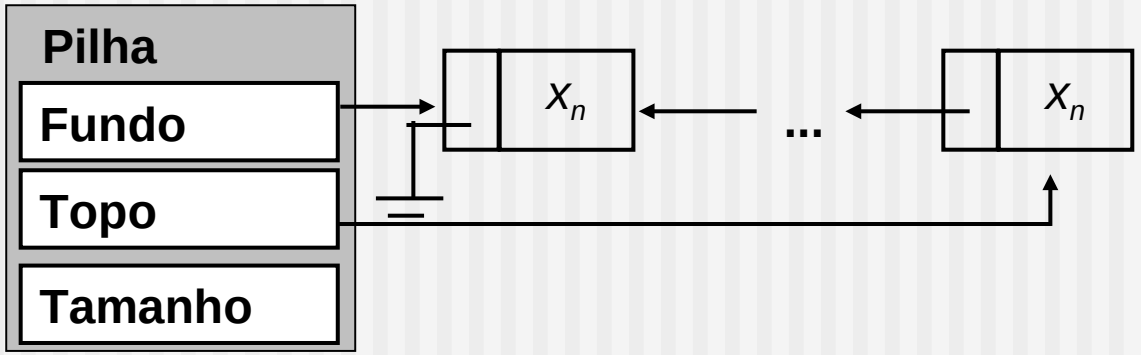
```
int Vazia(TipoPilha Pilha)
{
    return (Pilha.Topo == NULL);
}
```

```
//com célula cabeça
int Vazia(TipoPilha Pilha)
{
    return (Pilha.Topo == Pilha.Fundo);
}
```

# Implementação

```
void Empilha(TipoItem x, TipoPilha *Pilha)
{
    struct Celula *Aux;
    Aux = (struct Celula*) malloc(sizeof(Celula));
    Aux->Item = x;
    if(Vazia(*Pilha))
          Pilha->Fundo=Aux;
    Aux->Prox = Pilha->Topo;
    Pilha->Topo = Aux;
    Pilha->Tamanho++;

}
```



# Implementação

```
void Desempilha(TipoPilha *Pilha, TipoItem *Item)
{
    struct Celula *q;
    if (Vazia(*Pilha))
    {
          printf(" Erro   lista vazia\n");
          return;
    }
    q = Pilha->Topo;
    Pilha->Topo = q->Prox;
    *Item = q->Item;
    free(q);
    Pilha->Tamanho--;
    if(Pilha->Tamanho==0)
          Pilha->Fundo=Pilha->Topo;
}
```

# Exercício

Utilizando somente operações de empilhar e desempilhar, escreva um programa que remove um item com chave c fornecida pelo usuário da pilha.
Ao final da execução da função, a pilha deve ser igual à original, exceto pela ausência do item removido.