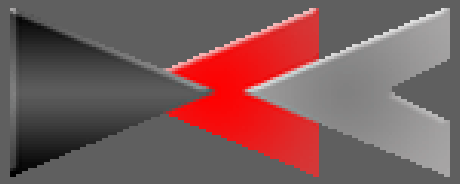

# DCC107



# Laboratório de Programação II

## Tipos Abstratos de Dados Pilhas e Filas (TAD Pilha e TAD Fila) em C

# Pilhas e Filas



- Pilhas e filas são casos especiais de listas lineares;
- Ambas possuem regras rigorosas para acessar os dados armazenados nelas;
- As operações de recuperação de dados são destrutivas, ou seja, para se alcançar dados intermediários a tais estruturas, é necessário destruir sequencialmente os dados anteriores.

# Pilhas



- É uma das estruturas de dados mais simples;
- É a estrutura de dados mais utilizada em programação, sendo inclusive implementada diretamente pelo *hardware* da maioria das máquinas modernas;
- A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo;
- Assim, quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo;
- A idéia desta estrutura é a de uma pilha de pratos.

# Pilhas



- Aplicações:
  - Verificação de parênteses;
  - Retirada de vagões de um trem;
  - Retirada de mercadorias em um caminhão de entregas;
  - Conversão de um número na base 10 para outra base numérica;

# Pilhas

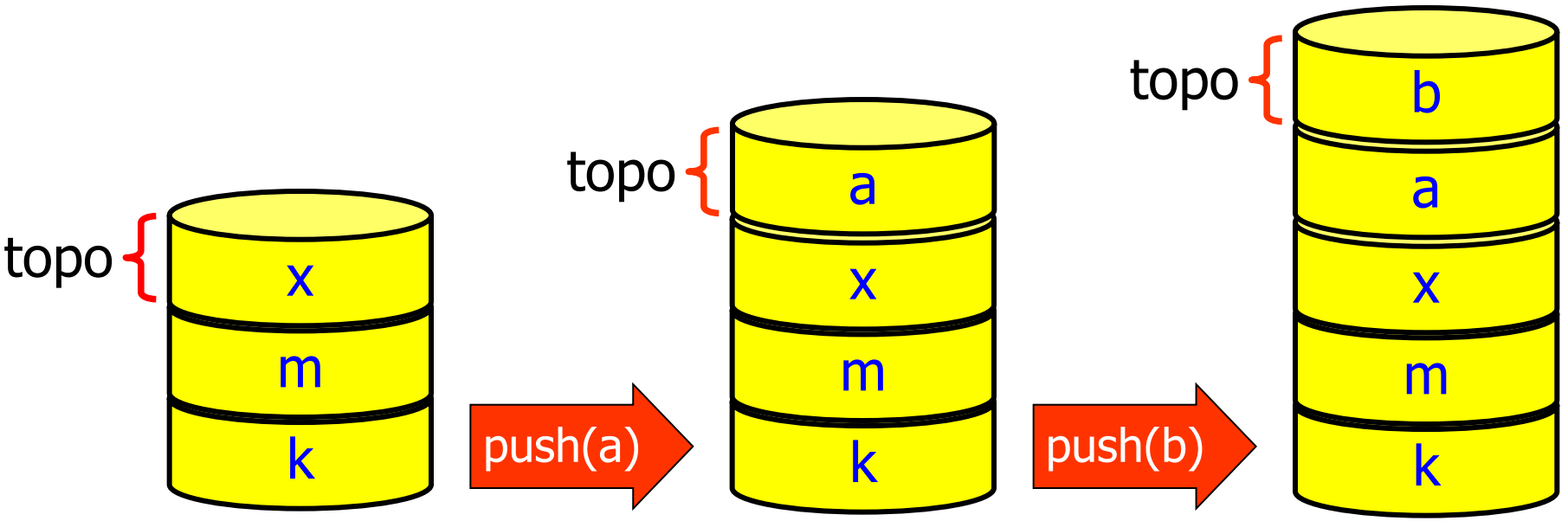


- Os elementos da pilha são retirados na ordem inversa à ordem em que foram introduzidos, o primeiro que sai é o último que entrou (LIFO – *Last In First Out*);
- Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:
  - **Empilhar** (***push***) um novo elemento, inserindo-o no topo;
  - **Desempilhar** (***pop***) um elemento, removendo-o do topo.

# Pilhas



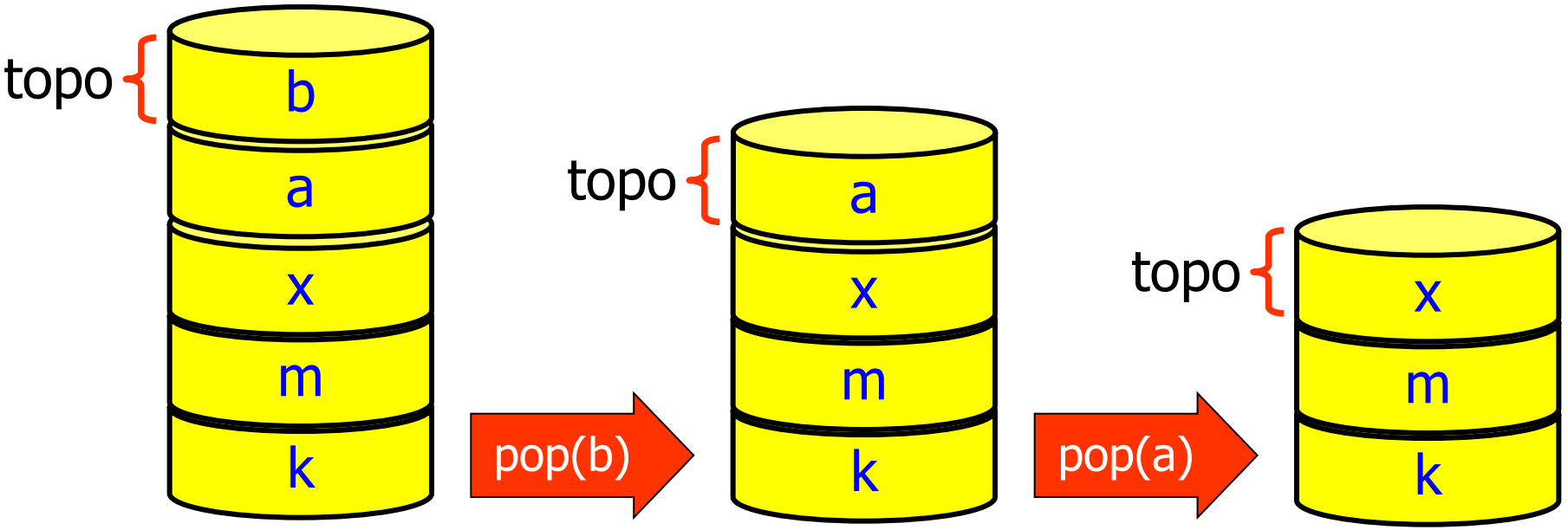
- Empilhar (*push*):



# Pilhas



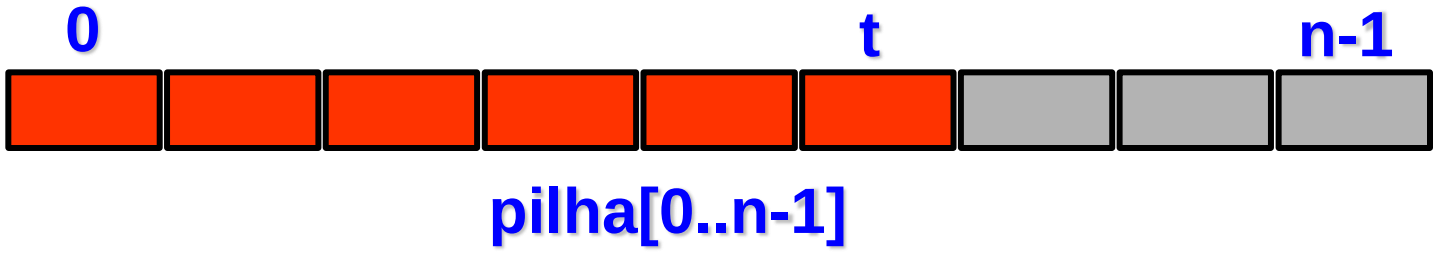
- Desempilhar (*pop*):



# Pilhas



- Vetor:
  - Supondo a pilha está armazenada em um vetor pilha[0..n-1];
  - Considerando que os elementos são inteiros (isso é só um exemplo, os elementos de pilha poderiam ser quaisquer outros objetos);
  - A parte do vetor ocupada pela pilha será:



# Pilhas



- Vetor:

```
#define MAX 50

struct pilha {
    int n;
    int vet[MAX];
};
```

# Pilhas



- Estrutura:

```
struct no {
    int info;
    struct no *prox;
};
typedef struct no No;

struct pilha {
    No* topo;
};
typedef struct pilha Pilha;
```

# Pilhas



- Operações básicas:
  - Criar uma estrutura de pilha;
  - Inserir um elemento no topo (*push*);
  - Remover o elemento do topo (*pop*);
  - Verificar se a pilha está vazia;
  - Liberar a estrutura de pilha.

# Pilhas



- Cria:

```
Pilha* cria()
{
    Pilha *p;
    p = (Pilha*) malloc(sizeof(Pilha));
    p->topo = NULL;
    return p;
}
```

- Vazia:

```
int vazia(Pilha *p)
{
    return (p->topo == NULL);
}
```

# Pilhas



- Inserir um elemento (*push*):

```
void empilha(Pilha *p, int v)
{
    No* aux;
    aux = (No*) malloc(sizeof(No));
    aux->info = v;
    aux->prox = p->topo;
    p->topo = aux;
}
```

# Pilhas



- Remover um elemento (*pop*):

```
int desempilha(Pilha *p)
{
    int v;
    No* aux;
    if(vazia(p))
    {
        printf("Pilha vazia.");
        exit(1);
    }
    v = p->topo->info;
    aux = p->topo;
    p->topo = aux->prox;
    free(aux);
    return v;
}
```

# Pilhas



- Libera:

```
void libera(Pilha *p)
{
    No* q = p->topo;
    while(q != NULL)
    {
        No *t = q->prox;
        free(q);
        q = t;
    }
    free(p);
}
```

# Filas



- São listas lineares que adotam a política FIFO (*First In First Out* – o primeiro que entra é o primeiro que sai) para a manipulação de elementos;
- As inserções são feitas no final da fila;
- As remoções são feitas no início da fila;
- A consulta na fila é feita desenfileirando elemento a elemento até encontrar o elemento desejado ou chegar ao final da fila;
- Esta estrutura é equivalente a uma fila de banco.

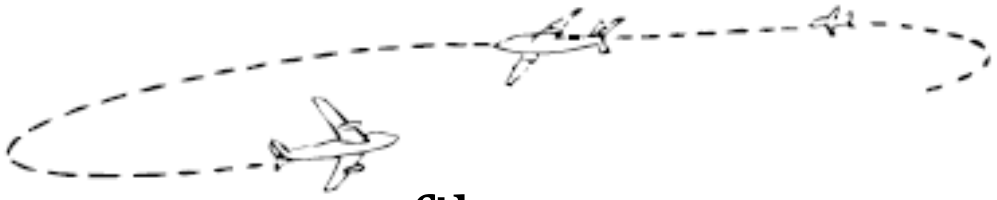
# Filas



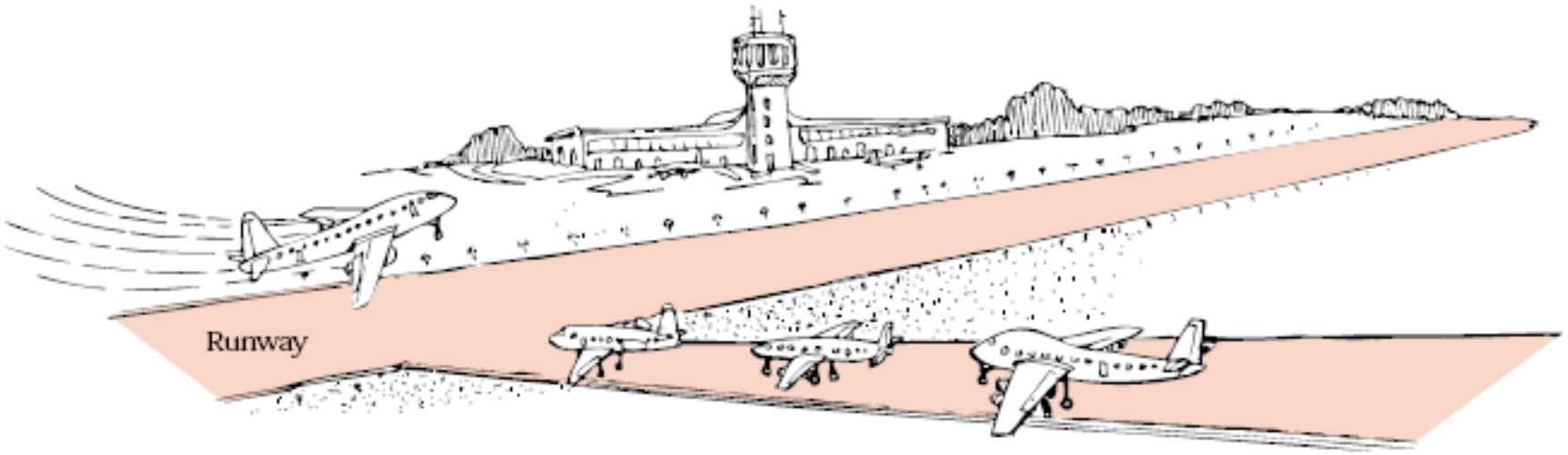
- Aplicações:
  - Alocação de recursos para impressão de documentos em uma impressora (*spooler* de impressão);
  - Atendimento de processos requisitados ao um Sistema Operacional;
  - Ordenação do encaminhamento dos pacotes em um roteador;
  - *Buffer* para gravação de dados em mídia.

# Filas





fila para pouso

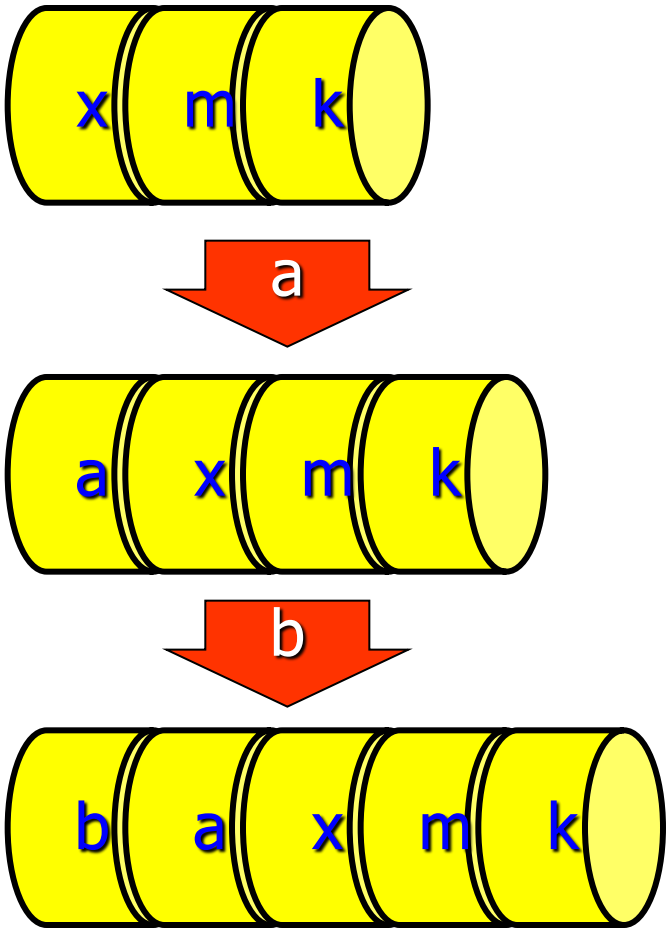



fila para decolagem

# Filas



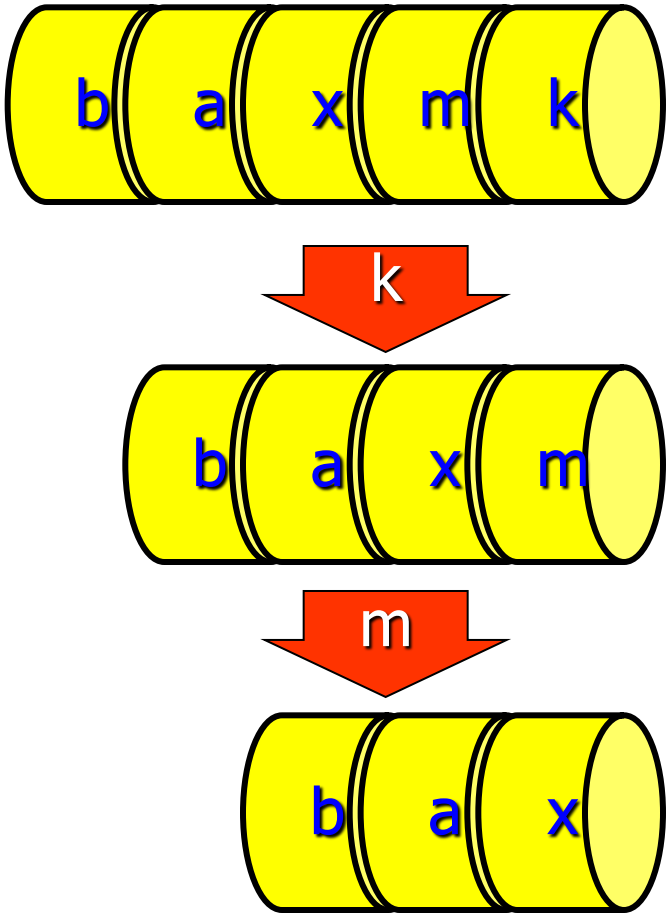
- Inserção:



# Filas



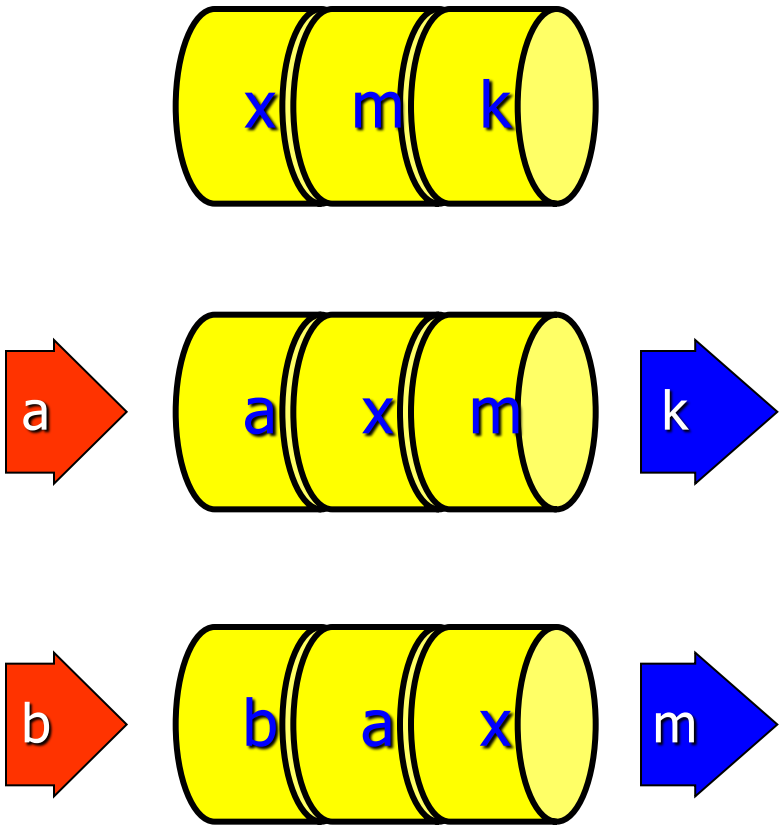
- Remoção:



# Filas



- Tamanho fixo:



# Filas



- Vetor:

```
#define MAX 100

struct fila {
    int inicio, fim;
    int vet[MAX];
};
```

# Filas



- Estrutura:

```
struct no {
    int info;
    struct no *prox;
};
typedef struct no No;

struct fila {
    No* inicio;
    No* fim;
};
typedef struct fila Fila;
```

# Filas



- Operações básicas:
  - Criação;
  - Destruição;
  - Inserção de um elemento;
  - Remoção de um elemento;
  - Localização de um elemento para consulta ou alteração;
  - Intercalação de filas;
  - Concatenação de filas;
  - Divisão de uma fila em duas.

# Filas



- Cria:

```
Fila* cria()
{
    Fila* f = (Fila*) malloc(sizeof(Fila));
    f->inicio = f->fim = NULL;
    return f;
}
```

- Vazia:

```
int vazia(Fila *f)
{
    return (f->inicio == NULL);
}
```

# Filas



- Inserir um elemento:

```
void enfileira(Fila *f, int v)
{
    No *q = (No*) malloc(sizeof(No));
    q->info = v;
    q->prox = NULL;
    if(vazia(f))
        f->inicio = q;
    else
        f->fim->prox = q;
    f->fim = q;
}
```

# Filas



- Remover um elemento:

```
int desenfileira(Fila *f)
{
    int v;
    No *q;
    if(vazia(f))
    {
        printf("Fila vazia.");
        exit(1);
    }
    q = f->inicio;
    v = q->info;
    f->inicio = f->inicio->prox;
    if(f->inicio == NULL)
        f->fim = NULL;
    free(q);
    return v;
}
```

# Filas



- Libera:

```
void libera (Fila* f)
{
    No* q = f->inicio;
    while(q!=NULL)
    {
        No* t = q->prox;
        free(q);
        q = t;
    }
    free(f);
}
```

# Exercícios



1. Criar operações para:
   a) Imprimir uma pilha;
   b) Imprimir uma fila;
   c) Pesquisar um elemento em uma pilha;
   d) Pesquisar um elemento em uma fila.
   e) Inverter uma pilha;
   f) Inverter uma fila;
2. Criar um TAD Pilha utilizando a representação vetorial, criando suas operações.
3. Criar um TAD Fila utilizando a representação vetorial, criando suas operações.