AED - Algoritmos e Estruturas de Dados

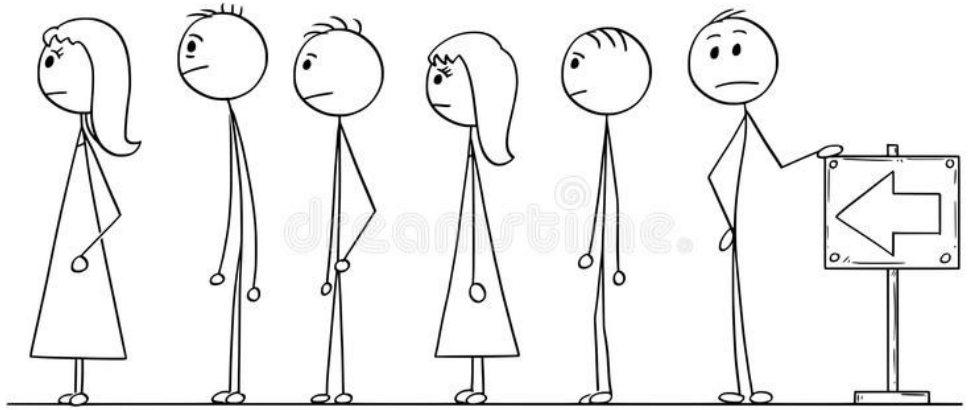
Aula 11 – Pilhas, Filas e Deques

Prof. Rodrigo Mafort

Pilhas, Filas e Deques

- Ao modificar o comportamento dos operadores de inserção e remoção, podemos estabelecer novas estruturas: pilhas, filas e deques.

Pilha



Fila

Pilhas

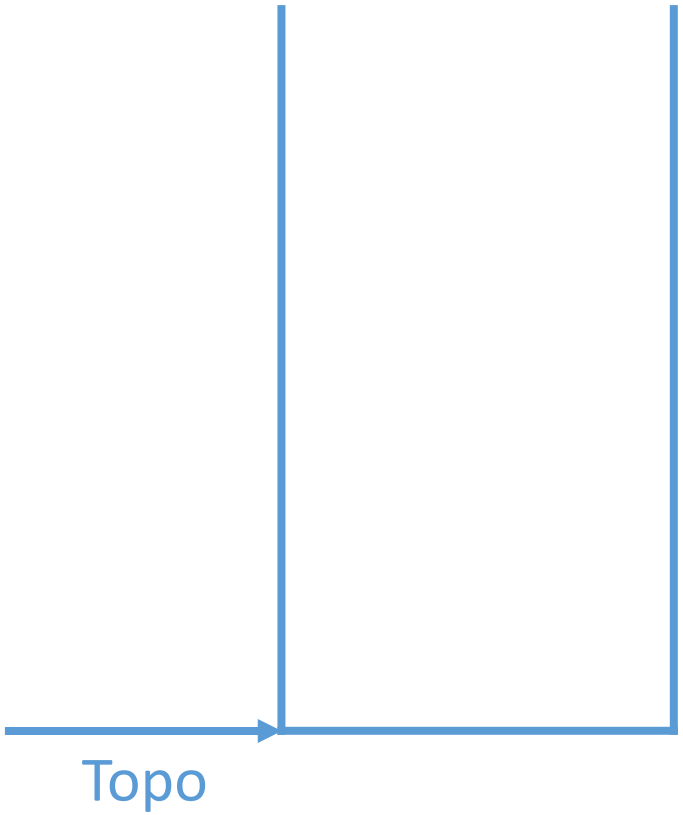
- A ideia da pilha é definir a ordem em que os elementos são inseridos e removidos.
- Nas pilhas, o último elemento inserido deve ser o primeiro removido
- Esse comportamento é chamado de LIFO ou UEPS
- **L**ast **I**n – **F**irst **O**ut ou **Ú**ltimo a **E**ntrar – **P**rimeiro a **S**air

Pilhas

- Nas pilhas precisamos controlar somente uma única informação: o **topo** da pilha.
- Em uma pilha, todas as operações são realizadas no topo da pilha.
- Um novo elemento é inserido após o último elemento da pilha
- Nas remoções, o último elemento é removido e retornado.

Pilhas

- Nas pilhas precisamos controlar um único atributo: o **topo** da pilha.
- Em uma pilha, todas as operações são realizadas no topo da pilha.
- Um novo elemento é inserido após o último elemento da pilha
- Nas remoções, o último elemento é removido e retornado.

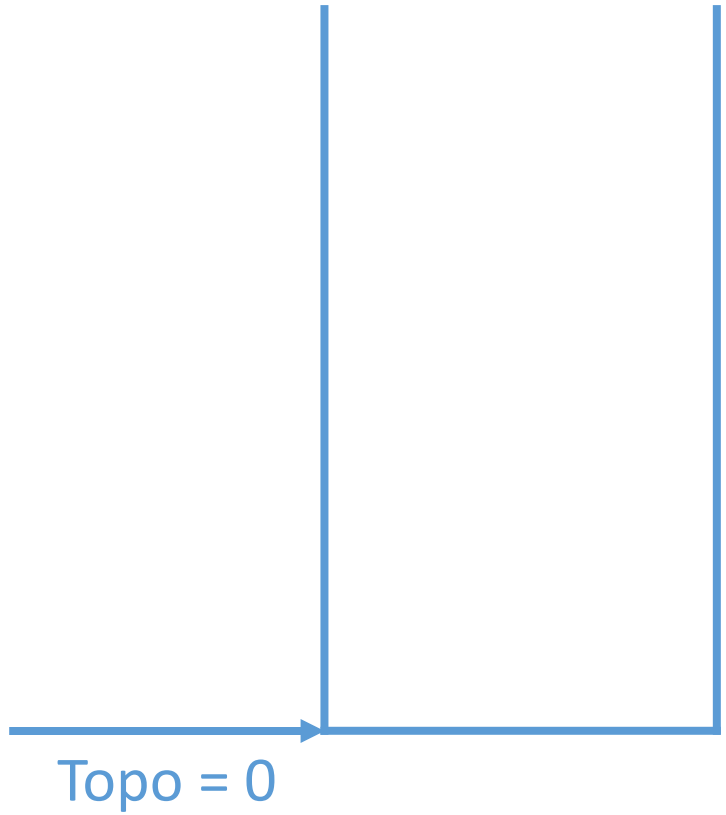


Pilhas – Inserir ou Empilhar

Empilhar("Harry Potter")

- Um novo elemento é inserido após o último elemento da pilha
- O último elemento é chamado de Topo
- Quando a pilha está vazia, topo = 0

- Empilhar
  1. Verificar se a pilha não está cheia (se for o caso)
  2. Adicionar o elemento na posição topo da lista
  3. Avançar topo em uma posição

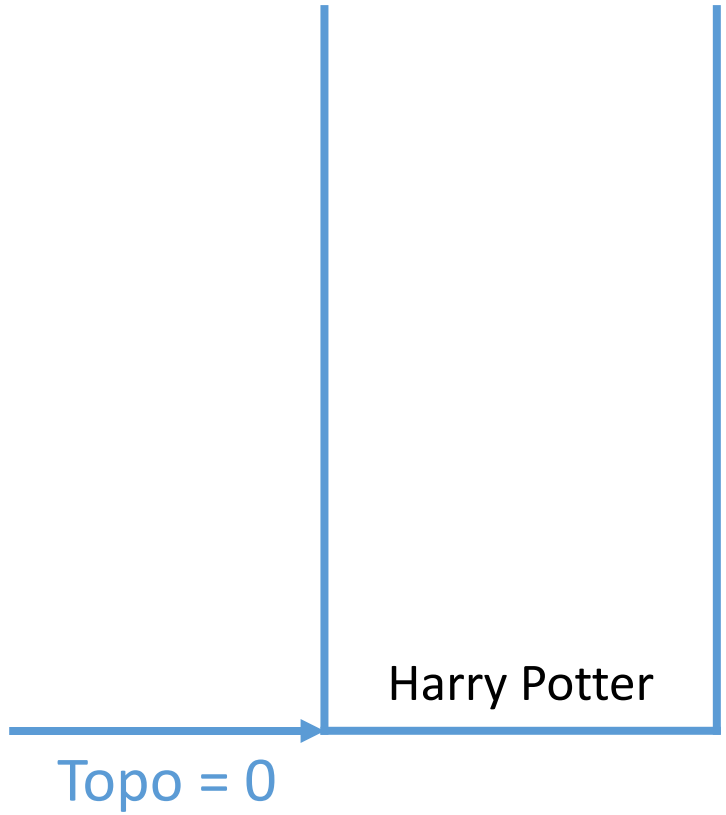


Pilhas – Inserir ou Empilhar

- Um novo elemento é inserido após o último elemento da pilha
- O último elemento é chamado de Topo
- Quando a pilha está vazia, topo = 0

- Empilhar
  1. Verificar se a pilha não está cheia (se for o caso)
  2. Adicionar o elemento na posição topo da lista (na posição topo da lista)
  3. Avançar topo em uma posição

Empilhar("Harry Potter")

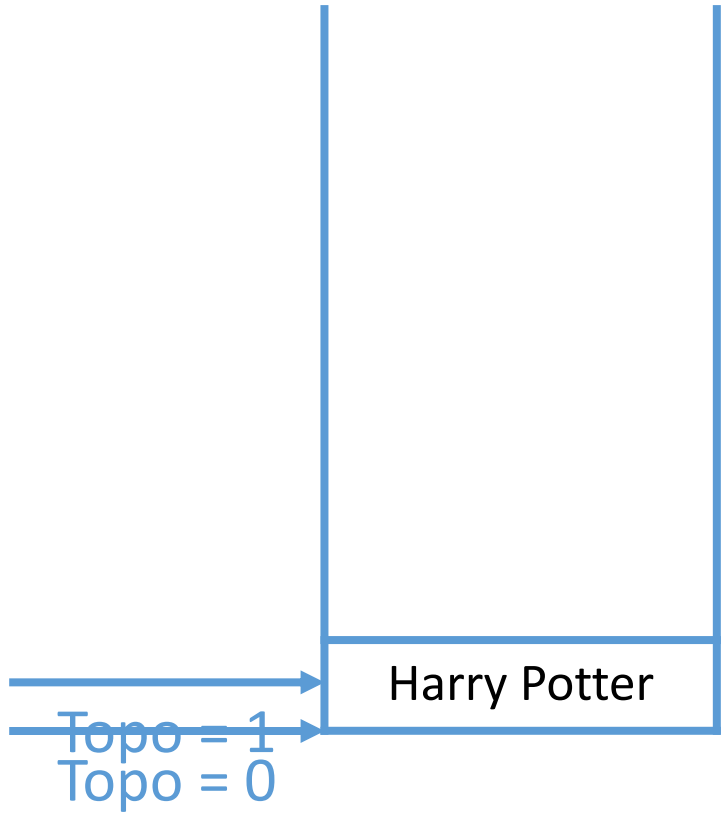


Pilhas – Inserir ou Empilhar

- Um novo elemento é inserido após o último elemento da pilha
- O último elemento é chamado de Topo
- Quando a pilha está vazia, topo = 0

- Empilhar
  1. Verificar se a pilha não está cheia (se for o caso)
  2. Adicionar o elemento na posição topo da lista (na posição topo da lista)
  3. Avançar topo em uma posição

Empilhar("Harry Potter")



Pilhas – Inicializar Pilha Vazia

- Assim como nas listas, precisamos de um vetor (estrutura subjacente) para armazenar a pilha.

- Podemos definir um vetor estaticamente ou podemos usar o malloc. Entretanto, independentemente da forma escolhida é necessário lembrar que o tamanho da lista é finito e que as posições da lista ocupam posições contiguas na memória.

Pilhas – Inicializar Pilha Vazia

```
#define TAM 50

typedef struct {
    int ID;
    char nome[255];
} elemento;

elemento P[TAM];
int topo = 0;
```

Definição de uma **constante** que especifica o tamanho máximo da pilha

Definição da estrutura utilizada para cada célula da pilha.

Alocação estática da pilha P

Observe que a pilha terá no máximo TAM elementos.

A pilha foi inicializada vazia (topo = 0).

Pilhas – Empilhar (Inserir)

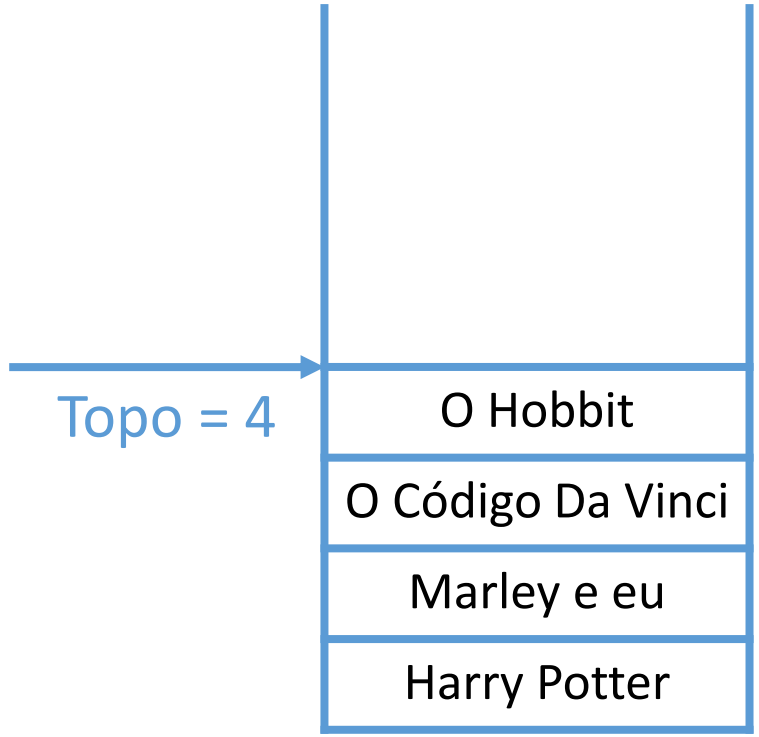
```
int Empilhar(elemento P[], int *topo)
{
    if (*topo >= TAM) //Pilha Cheia
        return 0;

    //Inserir na posição *topo da pilha P
    scanf("%d", &P[*topo].ID);
    //ler demais campos

    *topo = *topo + 1;
    return 1;
}
```

Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

Desempilhar()

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo

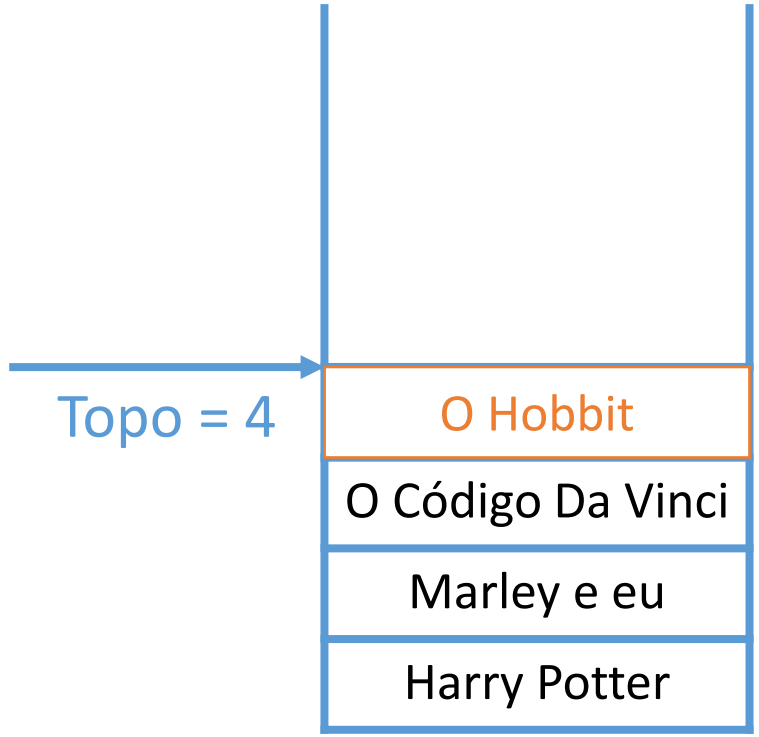


Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

Desempilhar()

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo

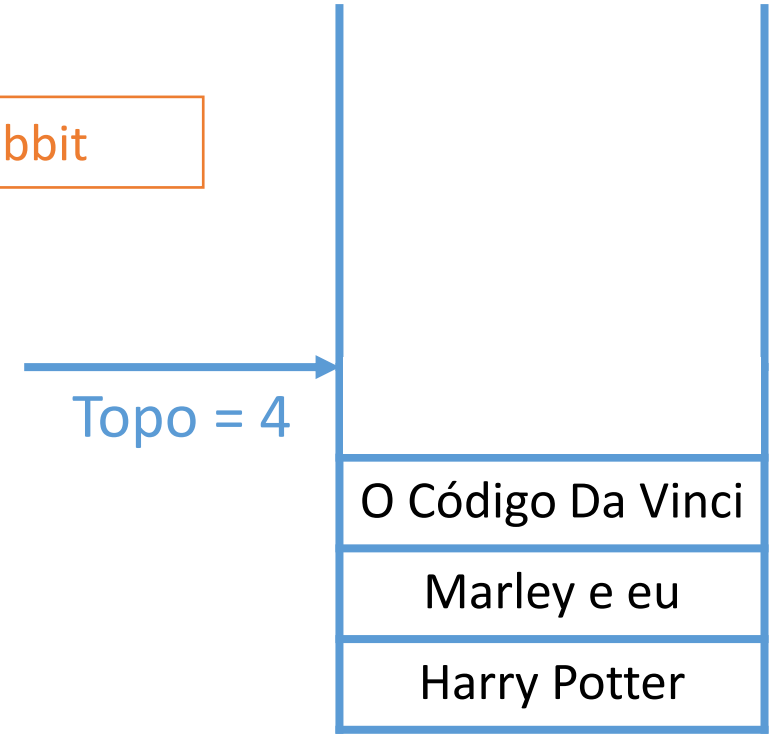


Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

Desempilhar()

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo

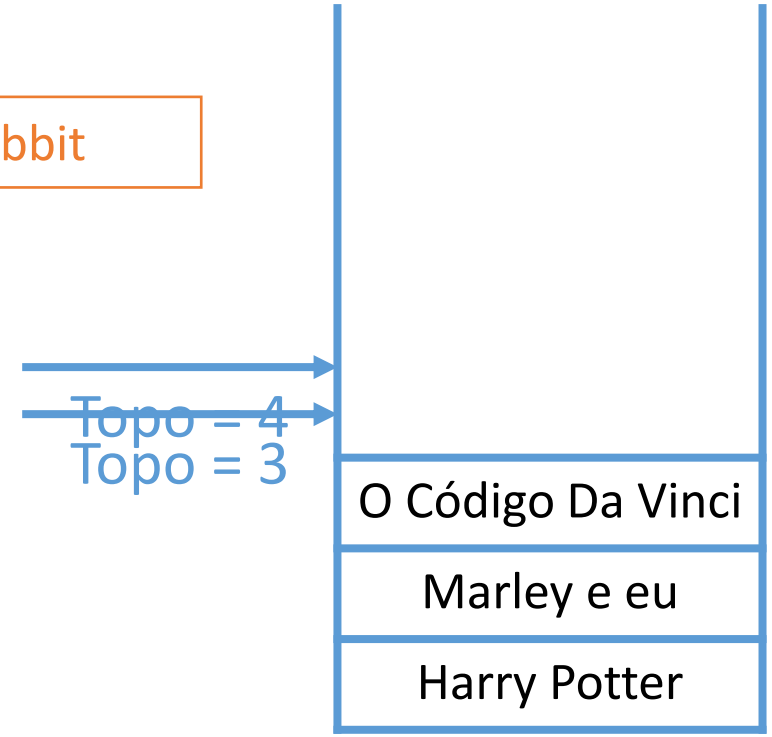


Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

Desempilhar()

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo



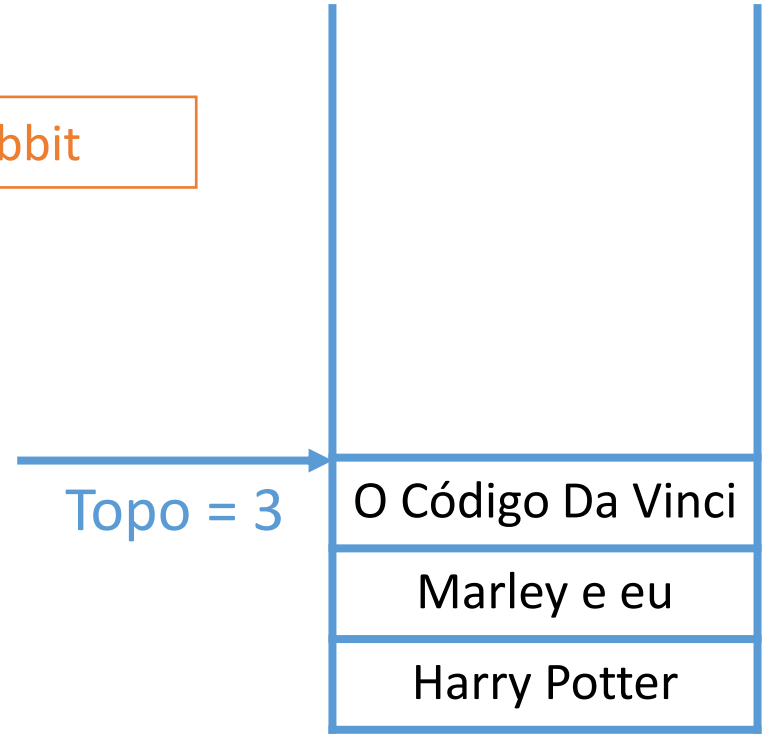
Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

Desempilhar()

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo

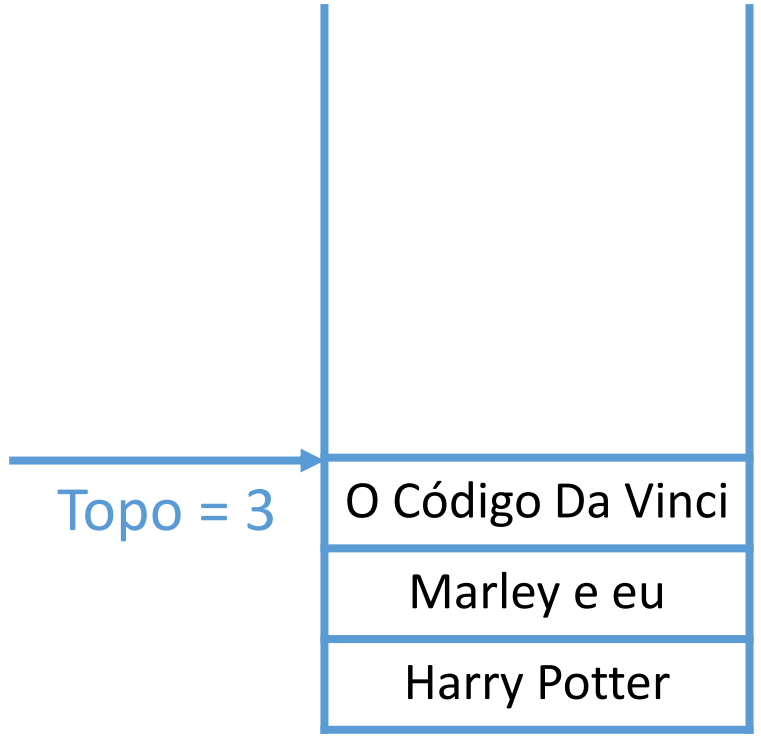
O Hobbit



Pilhas – Remover ou Desempilhar

- O último elemento inserido é removido e retornado
- O topo da pilha indica o último elemento inserido

- Desempilhar:
  1. Verificar se a pilha não está vazia
  2. Salvar o elemento na posição topo - 1
  3. Remover esse elemento da lista
  4. Decrementar o indicador do topo
  5. Retornar o elemento salvo



Pilhas – Desempilhar (Remover)

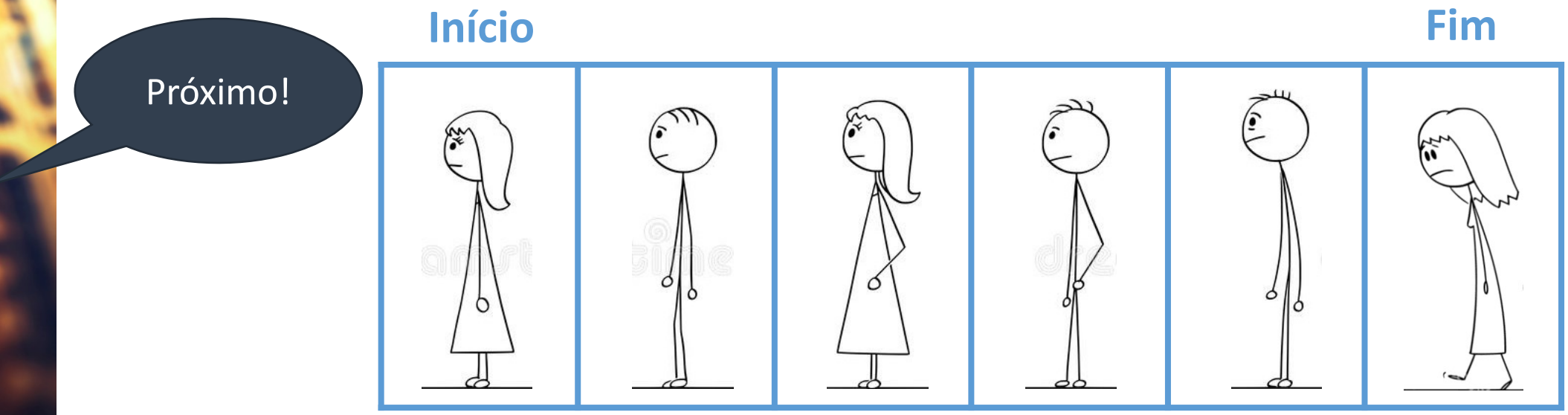
```
int Desempilhar(elemento P[], int *topo)
{
    if (*topo <= 0) //Pilha vazia
        return 0;

    printf("%d",P[*topo-1].ID);
    //Imprimir demais campos

    *topo = *topo - 1;
    return 1;
}
```

Filas

- A ideia de uma fila é estabelecer a ordem: o primeiro a entrar na fila deve ser o primeiro a ser atendido.
- Esse comportamento é chamado de FIFO ou PEPS
- **F**irst **I**n – **F**irst **O**ut ou **P**rimeiro a **E**ntrar – **P**rimeiro a **S**air

Filas

- A ideia de uma fila é estabelecer a ordem: o primeiro a entrar na fila deve ser o primeiro a ser atendido.
- Esse comportamento é chamado de FIFO ou PEPS
- **F**irst **I**n – **F**irst **O**ut ou **P**rimeiro a **E**ntrar – **P**rimeiro a **S**air



Filas

- Diferente das pilhas, as filas precisam manter três informações:
  - A estrutura subjacente (o vetor)
  - Uma variável para indicar onde é o início da fila
  - Uma variável para indicar onde é o final da fila

- Como veremos, existem duas implementações:
  - Uma considera que o início da fila é sempre a posição inicial do vetor. Mas isso implica que nas remoções, o espaço deixado na primeira posição precisa ser ocupado.
  - A outra considera que tanto o início quanto o final da fila são móveis dentro do vetor.

Filas

- Como visto na animação, precisamos manter controle sobre:
  - Quem é o primeiro da fila?
  - Quem é o último da fila?
  - A fila está cheia?
  - A fila está vazia?

- Vamos implementar um TAD para filas com 4 operações:
  - Inserir na fila (no final)
  - Remover da fila (no início)
  - Verificar se a fila está cheia
  - Verificar se a fila está vazia

Inserção e remoção na Fila

Enfileirar:

1. Verificar se a fila não está cheia
2. Inserir na última posição do vetor
3. Avançar o contador de final fila

Desenfileirar:

1. Verificar se a fila não está vazia
2. Remover e retornar o primeiro elemento da lista
3. Tratar o espaço em branco deixado pelo elemento removido

Filas – Primeira Implementação

- Essa primeira implementação considera que o início da fila (posição onde as remoções serão executadas) é sempre a primeira posição do vetor.
- Essa implementação possibilita verificar de forma rápida se a fila está vazia ou não: tem alguém na posição 0?
- As inserções também são sempre realizadas após a última posição ocupada da lista
- Entretanto, como as remoções deixam um espaço vazio na primeira posição, precisamos mover todos os demais elementos uma posição à frente – para “fechar” o espaço deixado pela remoção.
- Considerando que o início da fila é sempre a posição 0, não precisamos de uma variável para indicar essa posição

Filas – Inicializar Fila Vazia

```
#define TAM 50

typedef struct {
    int ID;
    char nome[255];
} elemento;

elemento F[TAM];
int Fim = 0;
```

Definição de uma **constante** que especifica o tamanho máximo da fila

Definição da estrutura utilizada para cada célula da fila.

Alocação estática da fila F

A pilha foi inicializada vazia:
Fim = 0

Filas – Enfileirar (Inserção)

```
int Enfileirar(elemento F[], int *fim)
{
   if (*fim >= TAM)  //Fila cheia
      return 0;

   //Inserir na posição *fim da fila F
   scanf("%d", &F[*fim].ID);
   //ler demais campos

   *fim = *fim + 1;
   return 1;
}
```

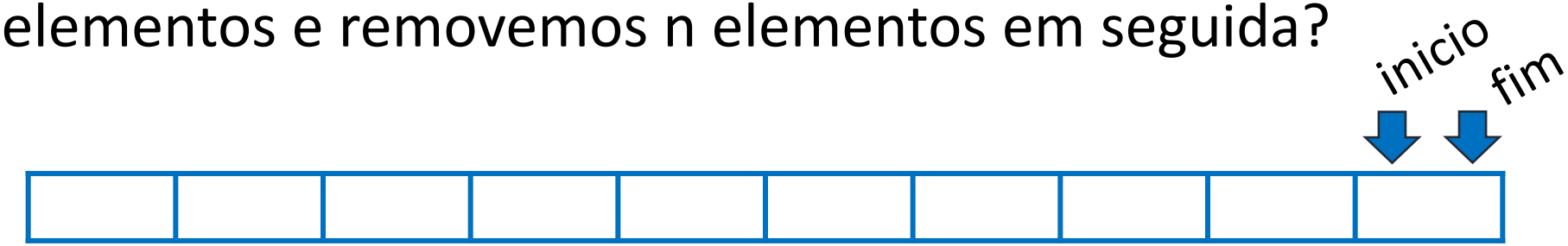
Filas – Desenfileirar (Remoção)

```
int Desenfileirar(elemento F[], int *fim)
{
    if (*fim <= 0) //Fila vazia
        return 0;

    printf("%d",F[0].ID); //Imprimir os demais campos

    for (int i = 0; i < *fim - 1; i++)
        F[i].ID = F[i+1].ID; //Copiar os demais campos

    *fim = *fim - 1;
    return 1;
}
```

Filas – Primeira Implementação

- Inserção: $O(1)$
- Remoção: $O(n)$
  - $O(1)$ para remover da primeira posição
  - $O(n)$ para deslocar todos os elementos uma posição à frente

- Como podemos reduzir o custo da remoção?
  - Vamos indicar onde a fila começa.
  - Essa estratégia tem um problema: O que acontece quando inserimos n elementos e removemos n elementos em seguida?



Filas – Segunda Implementação

- Essa implementação visa reduzir o custo associado as remoções e ainda lidar com os conflitos entre início da fila e final da fila.

- A ideia é considerar o vetor como um circulo. O final é conectado ao início.

Filas – Inicializar Fila Vazia

```
#define TAM 50

typedef struct {
    int ID;
    char nome[255];
} elemento;

elemento F[TAM];
int inicio = 0;
int nElementos = 0;
```

Definição de uma **constante** que especifica o tamanho máximo da fila

Definição da estrutura utilizada para cada célula da fila.

Alocação estática da fila F

A pilha foi inicializada vazia:
inicio = 0
nElementos = 0

Filas – Enfileirar (Inserção)

```
int Enfileirar(elemento F[], int *inicio, int* nElementos)
{
    if (*nElementos >= TAM)  //Fila cheia
        return 0;

    int fim = (*inicio + *nElementos) % TAM;

    scanf("%d", &F[fim].ID);
    //ler demais campos da posição fim (da fila)

    *nElementos = *nElementos + 1;
    return 1;
}
```

Filas – Desenfileirar (Remoção)

```
int Desenfileirar(elemento F[], int *inicio, int* nElementos)
{
    if (*nElementos <= 0) //Fila vazia
        return 0;

    printf("%d",F[*inicio].ID);
    //Imprimir demais campos da posição *inicio (da fila)

    *inicio = (*inicio + 1) % TAM;
    *nElementos = *nElementos - 1;

    return 1;
}
```

Filas - Complexidade

Operação	Primeira Implementação	Segunda Implementação
Inicialização	$O(1)$	$O(1)$
Verificação de Fila Vazia	$O(1)$	$O(1)$
Verificação de Fila Cheia	$O(1)$	$O(1)$
Inserção	$O(1)$	$O(1)$
Remoção	$\boldsymbol{O(n)}$	$\boldsymbol{O(1)}$

Deques

- Deques são estruturas que permitem a manipulação apenas pelos extremos (esquerdo e direito).

- Assim como nas filas, apenas o primeiro elemento de cada ponta pode ser removido.
- As inserções ocorrem ao final dos elementos inseridos em uma determinada ponta.
- O deque se torna cheio quando todas as posições estão ocupadas – quando isso ocorre, o fim das duas filas (direita e esquerda”) se encontram.

Deques

- Considerando as particularidades, os deques não admitem a implementação “circular” apresentada para as filas.

- Desta forma, as remoções tem custo $O(n)$, quando a estrutura subjacente é um vetor.

- Tente implementar um deque, considerando que:
  - Cada extremo se comporta exatamente como uma fila
  - Um deque está cheio quando o final das duas filas coincide
  - Vale observar que um lado pode estar vazio e outro, totalmente cheio.
  - As operações de inserção e remoção devem receber um parâmetro adicional que indica o extremo onde a operação deve ser realizada (esquerdo ou direito). Outra opção é implementar dois métodos de inserção e remoção (um para cada extremo).

Conteúdo da Disciplina

- Com essa aula fechamos a primeira parte da disciplina:
  - Complexidade Computacional
  - Estruturas de alocação sequencial
  - Algoritmos de Ordenação

- Na próxima aula teremos a primeira avaliação:
  - Dia 17/05/2024

- Em seguida, vamos iniciar a segunda parte do conteúdo:
  - Estruturas de alocação dinâmica (listas encadeadas, árvores e grafos)