Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

TAD – Pilhas

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- A **Pilha** pode ser entendida como uma lista linear em que todas as inserções, retiradas e, geralmente, todos os acessos são feitos em apenas um extremo da lista.

- Os itens são colocados um sobre o outro.
  - O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.

1. Introdução

- O modelo intuitivo é o de um monte de pratos em uma prateleira, sendo conveniente retirar ou adicionar pratos na parte superior.

- Como são as Pilhas?

  - Insere-se elementos no topo da pilha.

  - Remove-se ou utiliza-se apenas o elemento que estiver no topo da pilha.

1. Introdução

- Pilhas são casos especiais de listas.
  - Nas listas, quando precisávamos criar um novo elemento, poderíamos inseri-lo ou removê-lo **de qualquer posição da lista.**
  - Exemplos:
    - Na primeira posição;
    - Na última posição; ou
    - Em qualquer parte no meio da lista.

1. Introdução

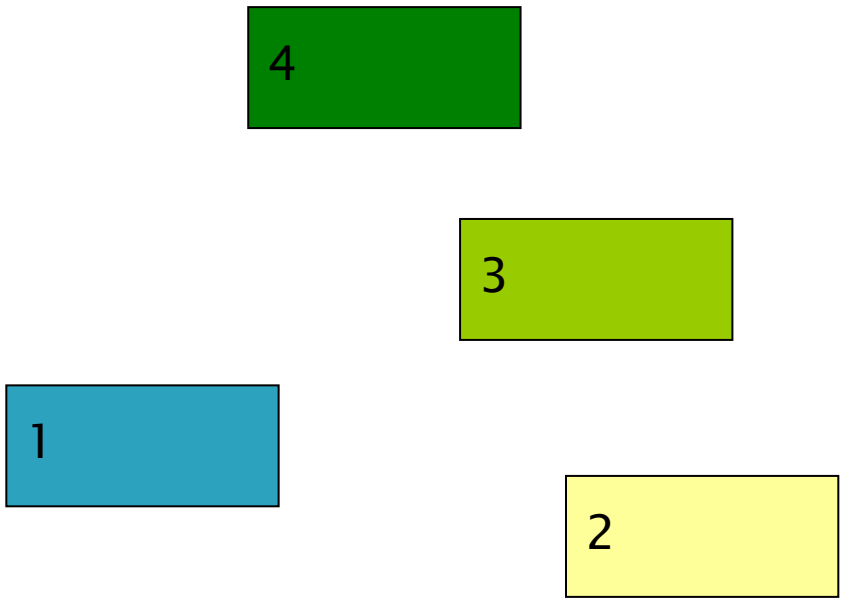
- Numa pilha existe uma regra básica a ser seguida:
  - **Último a chegar é o primeiro a sair**
  - Do inglês: ***LIFO*** – *Last In, First Out*
- Um novo elemento da pilha somente pode ser inserido no topo da pilha.
- Um elemento só pode ser removido do topo da pilha.

1. Introdução

- Existe uma ordem linear para pilhas, do “mais recente para o menos recente”.
- É ideal para processamento de estruturas aninhadas de profundidade imprevisível.
- Uma pilha contém uma sequência de obrigações adiadas.
  - A ordem de remoção garante que as estruturas mais internas serão processadas antes das mais externas.

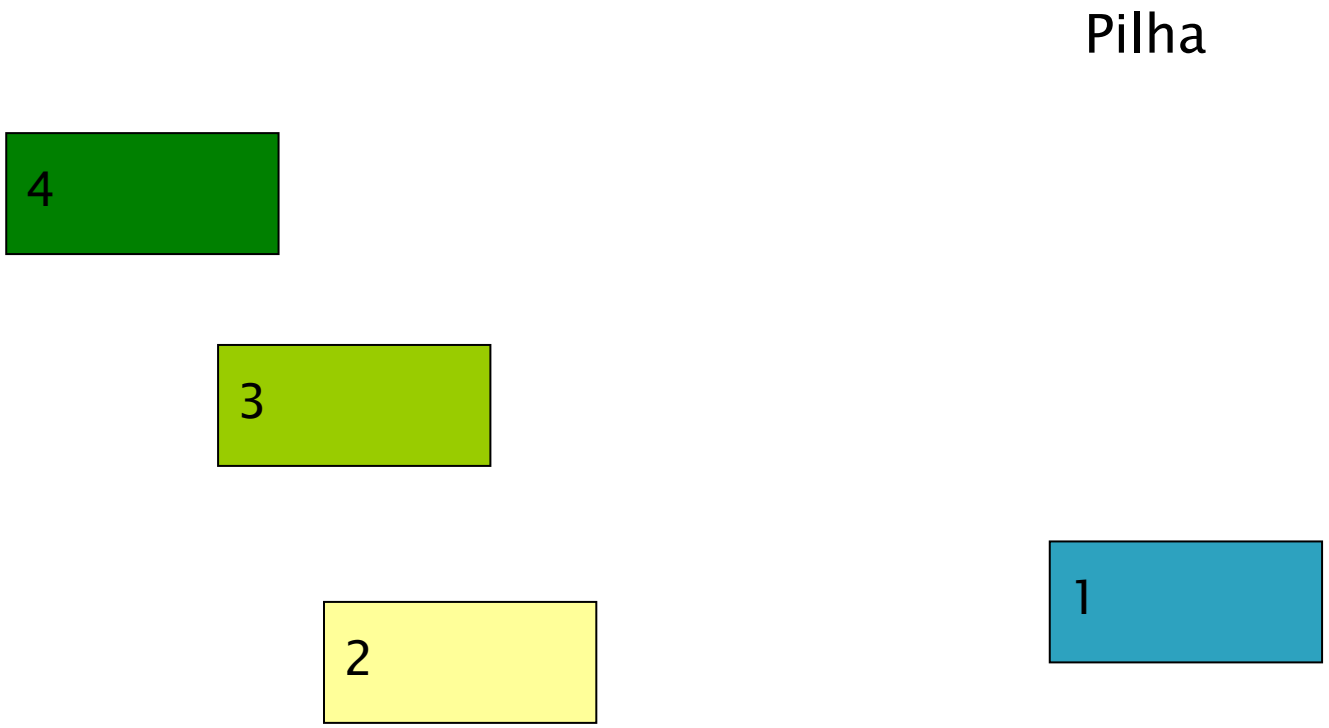
1. Introdução

- **Funcionamento da Pilha:** Pilha vazia



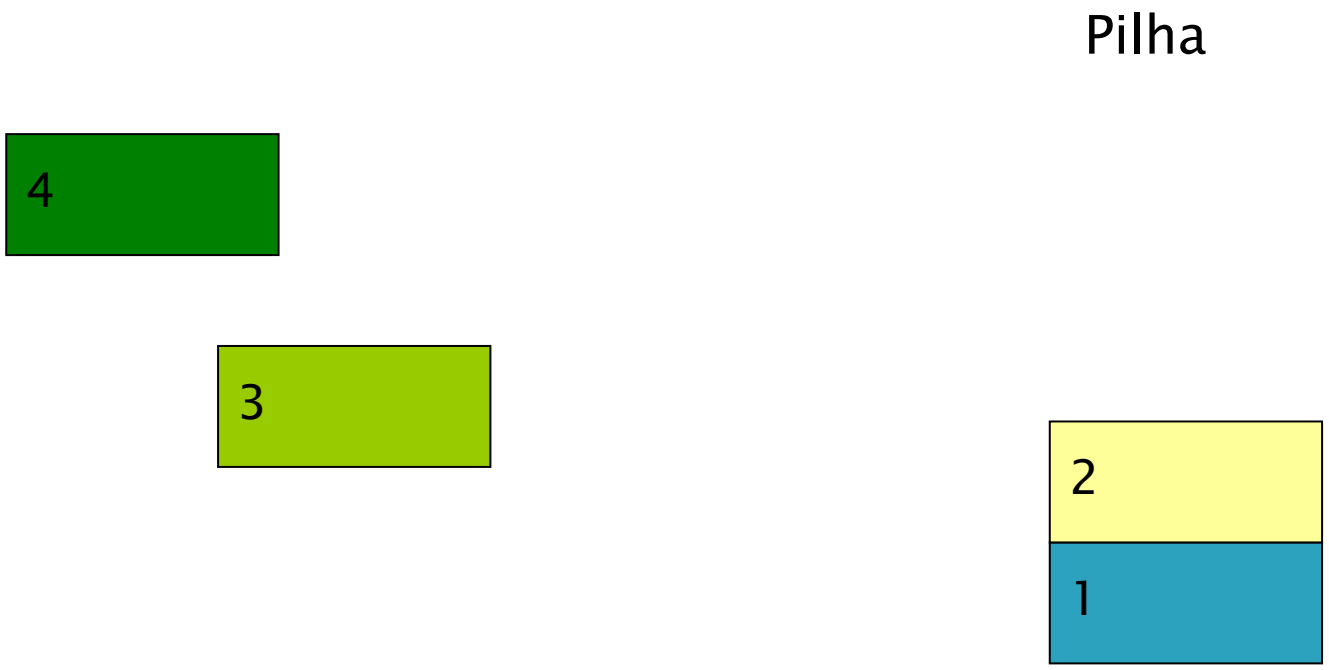
1. Introdução

- **Funcionamento da Pilha:** Empilhou



1. Introdução

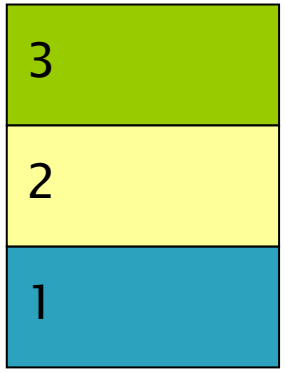
- **Funcionamento da Pilha:** Empilhou



1. Introdução

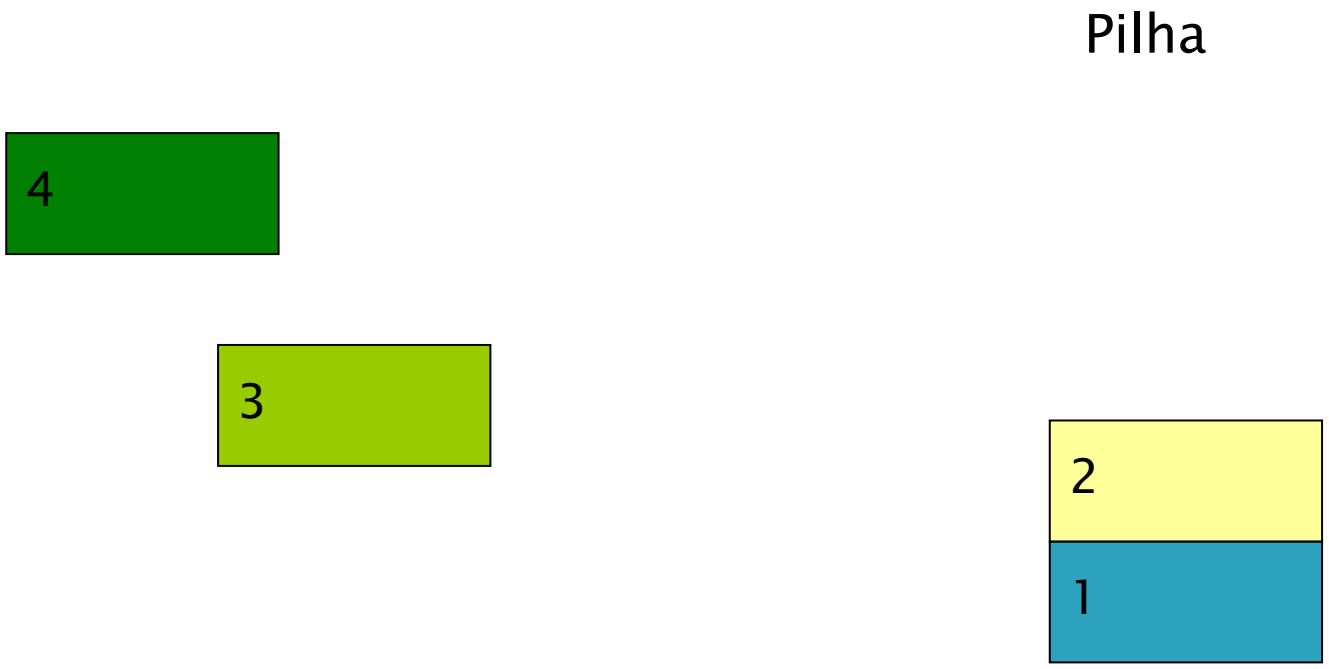
- **Funcionamento da Pilha:** Empilhou

Pilha



1. Introdução

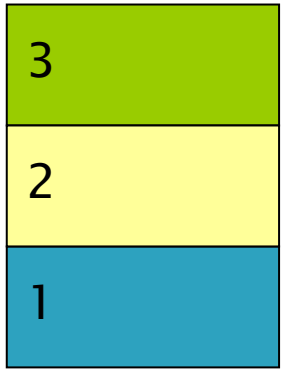
- **Funcionamento da Pilha:** Desempilhou



1. Introdução

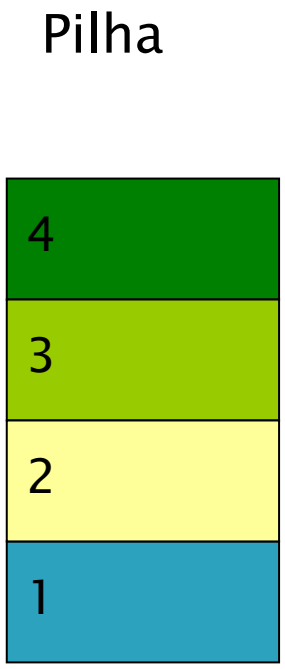
- **Funcionamento da Pilha:** Empilhou

Pilha



1. Introdução

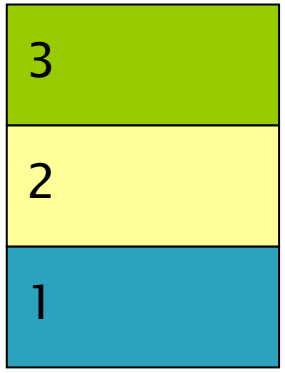
- **Funcionamento da Pilha:** Empilhou



1. Introdução

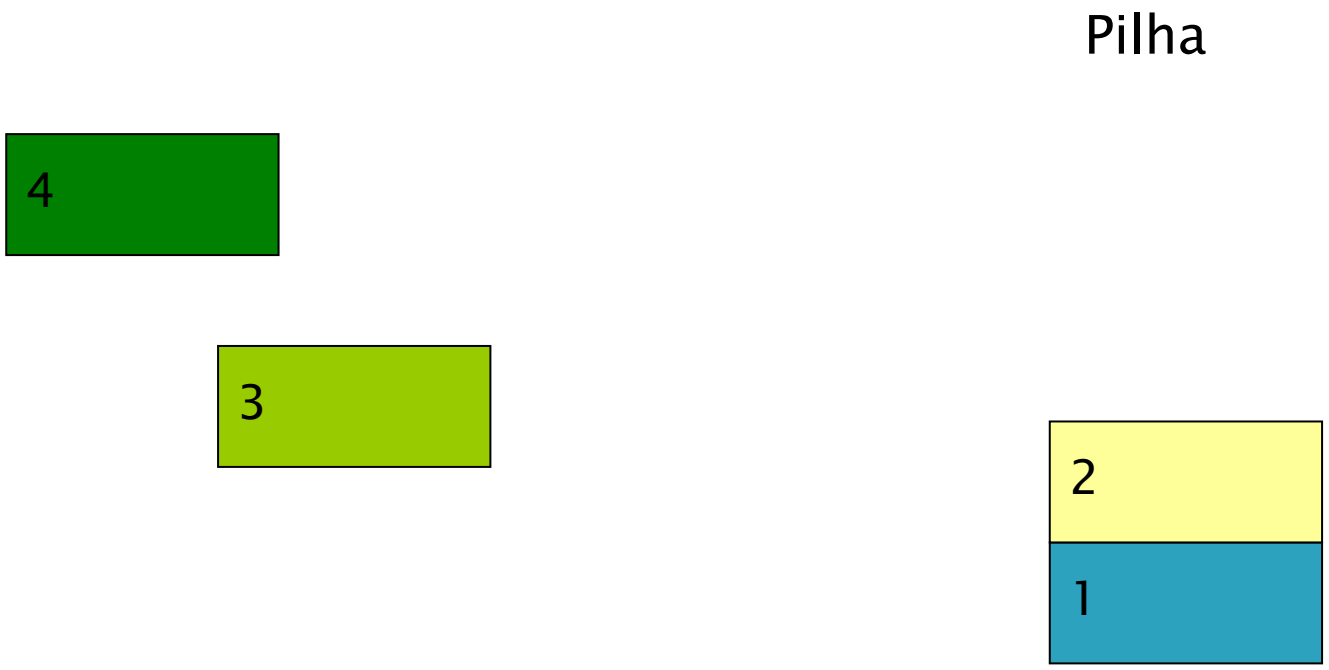
- **Funcionamento da Pilha:** Desempilhou

Pilha



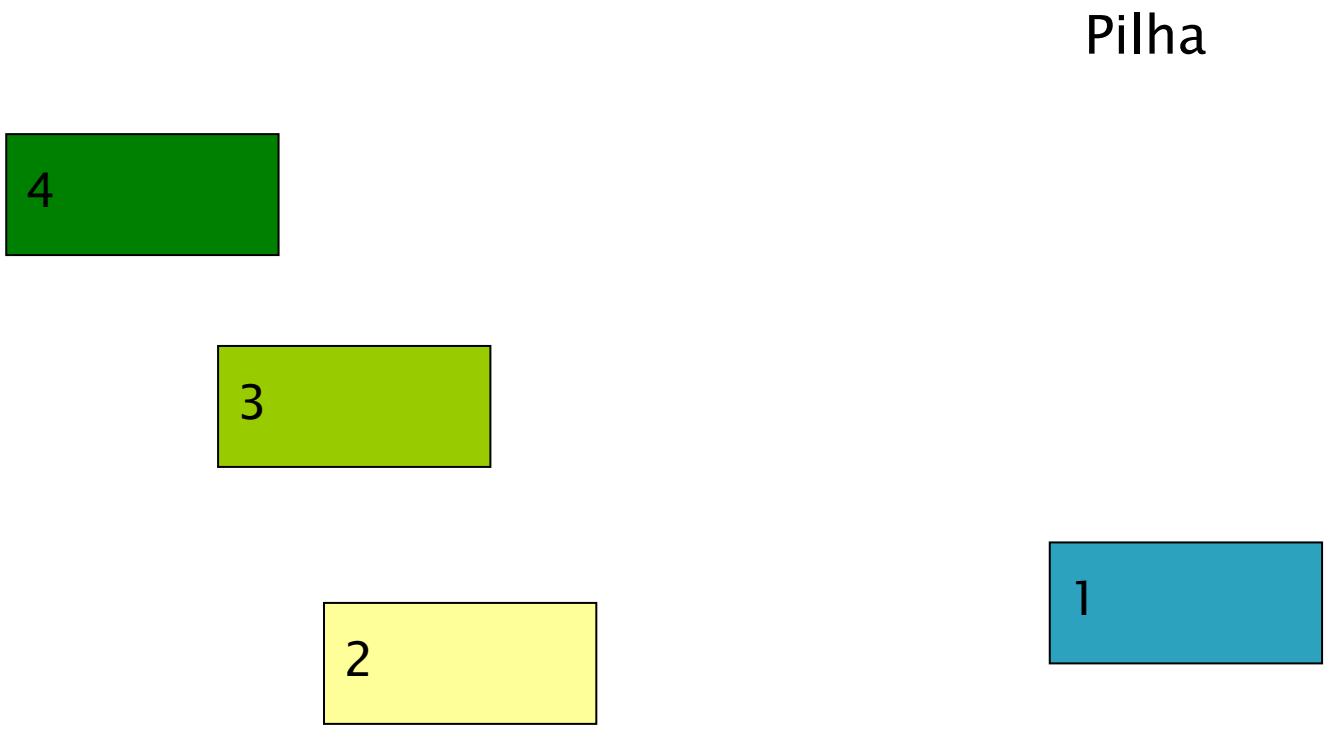
1. Introdução

- **Funcionamento da Pilha:** Desempilhou



1. Introdução

- **Funcionamento da Pilha:** Desempilhou



1. Introdução

- Aplicações em estruturas aninhadas:
  - Quando é necessário caminhar em um conjunto de dados e guardar uma lista de coisas a fazer posteriormente.
  - O controle de sequências de chamadas de subprogramas.
  - A sintaxe de expressões aritméticas.
- As pilhas ocorrem em estruturas de natureza recursiva (como árvores). Elas são utilizadas para implementar a **recursividade.**

2. TAD Pilha

- O que o TAD Pilha deveria conter?
  - Representação do tipo da pilha.
  - Conjunto de operações que atuam sobre a pilha.
- Quais operações deveriam fazer parte da pilha?
  - Depende de cada aplicação.
  - Mas, um conjunto padrão pode ser definido.

2. TAD Pilha

- Operações necessárias à grande maioria das aplicações:

FPVazia(Pilha): faz a pilha ficar vazia.

PVazia(Pilha): retorna ***true*** se a pilha está vazia; caso contrário, retorna ***false***.

Empilha(Pilha, x): insere o item **x** no topo da pilha.

Desempilha(Pilha, x): retorna o item x no topo da pilha, retirando-o da pilha.

Tamanho(Pilha): esta função retorna o número de itens da pilha.

2. TAD Pilha

- Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas.

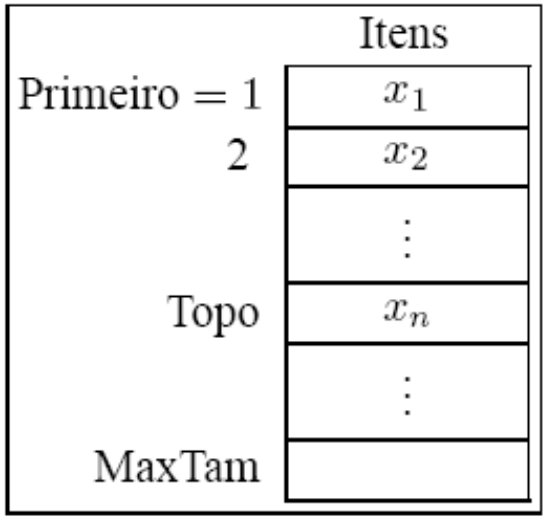
- As duas representações mais utilizadas são:

  - Implementação por **arrays** (vetores).

  - Implementação por **ponteiros**.

3. Implementação por Array

- Os itens são armazenados em posições contíguas de memória.
- Como as **inserções** e as **retiradas** ocorrem no topo da pilha, um campo chamado **topo** é utilizado para controlar a posição do item no topo da pilha.



3. Implementação por Array

- Estrutura:

```
#define MaxTam 1000

typedef int Tchave;

typedef struct {
  TChave Chave;
  /* outros componentes */
}TItem;

typedef struct {
  TItem vItem[MaxTam];
  int iTopo;
}TPilha;
```

3. Implementação por Array

```
void FPVazia (TPilha* pPilha) {
  pPilha->iTopo = 0;
}

int PVazia (TPilha* pPilha) {
  return (pPilha->iTopo == 0);
}

int PTamanho (TipoPilha* pPilha) {
  return (pPilha->iTopo);
}
```

3. Implementação por Array

```
int Empilha (TPilha* pPilha, TItem* pItem) {

  if (pPilha->iTopo == MAXTAM)
    return 0;

  pPilha->vItem[pPilha->iTopo] = *pItem;
  pPilha->iTopo++;
  return 1;
}
```

3. Implementação por Array

```
int Desempilha (TPilha* pPilha, TItem* pItem) {

  if (PVazia(pPilha))
    return 0;

  pPilha->iTopo--;
  *pItem = pPilha->vItem[pPilha->iTopo];
  return 1;
}
```

4. Implementação por Ponteiro

- Há uma célula **`Cabeça`** no topo para facilitar a implementação das operações **`empilha`** e **`desempilha`** quando a pilha está vazia.

- Para desempilhar o item x_n basta desligar a célula cabeça da lista e a célula que contém x_n passa a ser a célula cabeça.

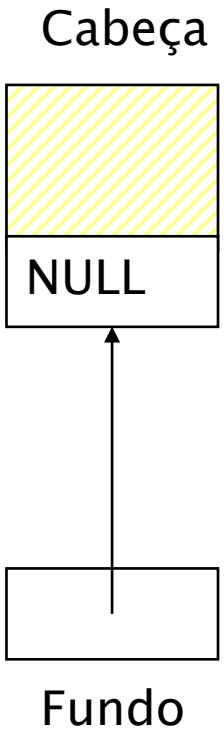
- Para empilhar um novo item, basta fazer a operação contrária, criando uma nova célula cabeça e colocando o novo item na antiga.

4. Implementação por Ponteiro

- O campo **`tamanho`** evita a contagem do número de itens na função **`Tamanho`**.
- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- O registro **`TPilha`** contém um apontador para o topo da pilha (célula **`cabeça`**) e um apontador para o fundo da pilha.

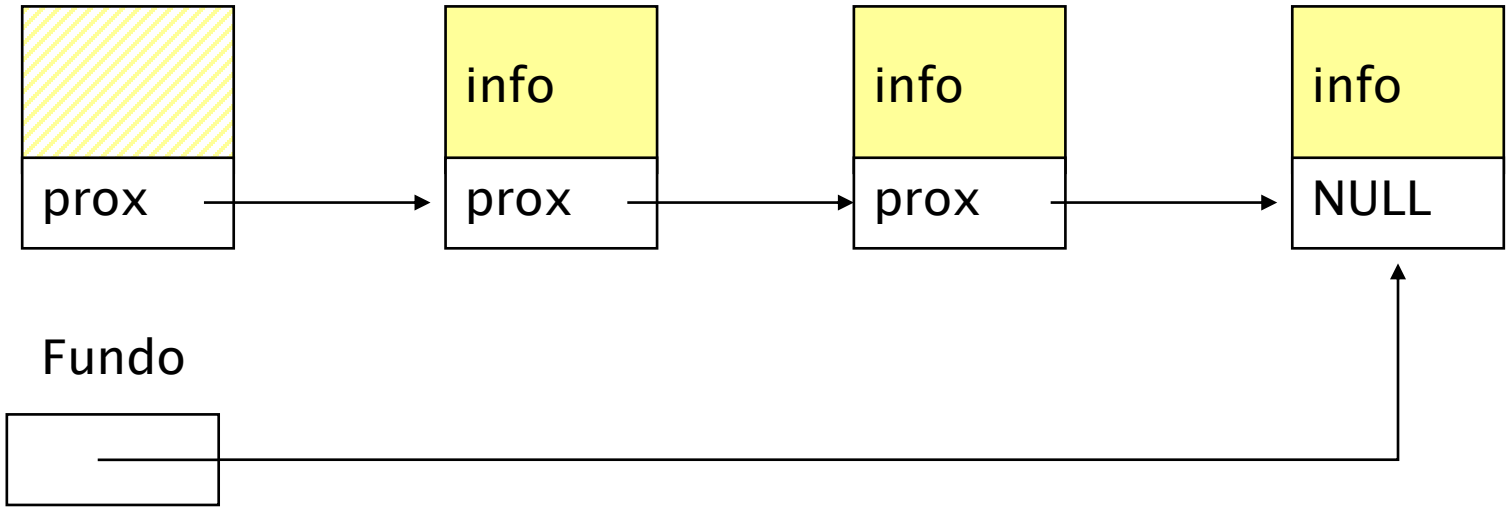
4. Implementação por Ponteiro

- Cria Pilha Vazia



4. Implementação por Ponteiro

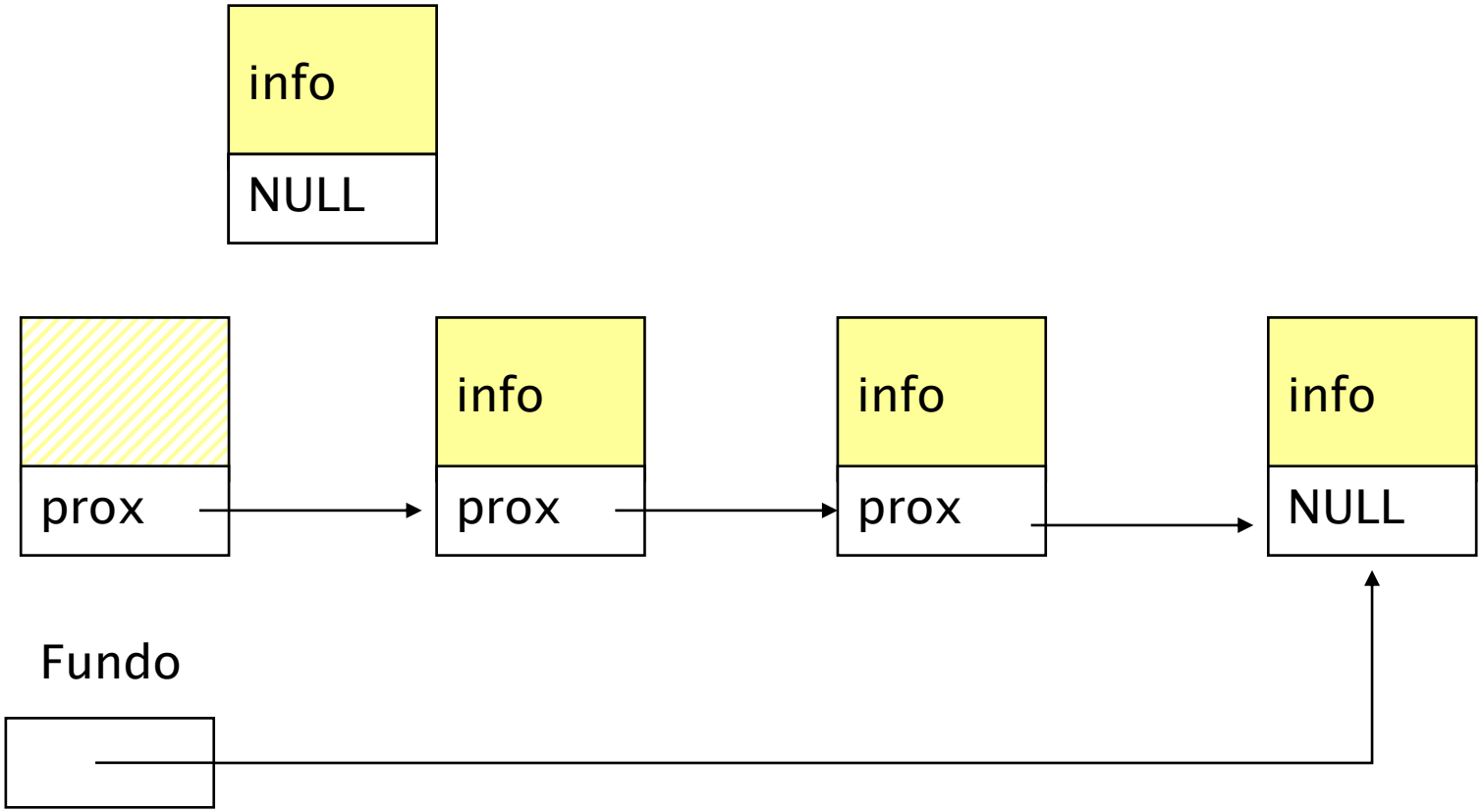
- Opção única de posição onde se pode **inserir**:



  - **Topo** da pilha, ou seja, **primeira** posição.

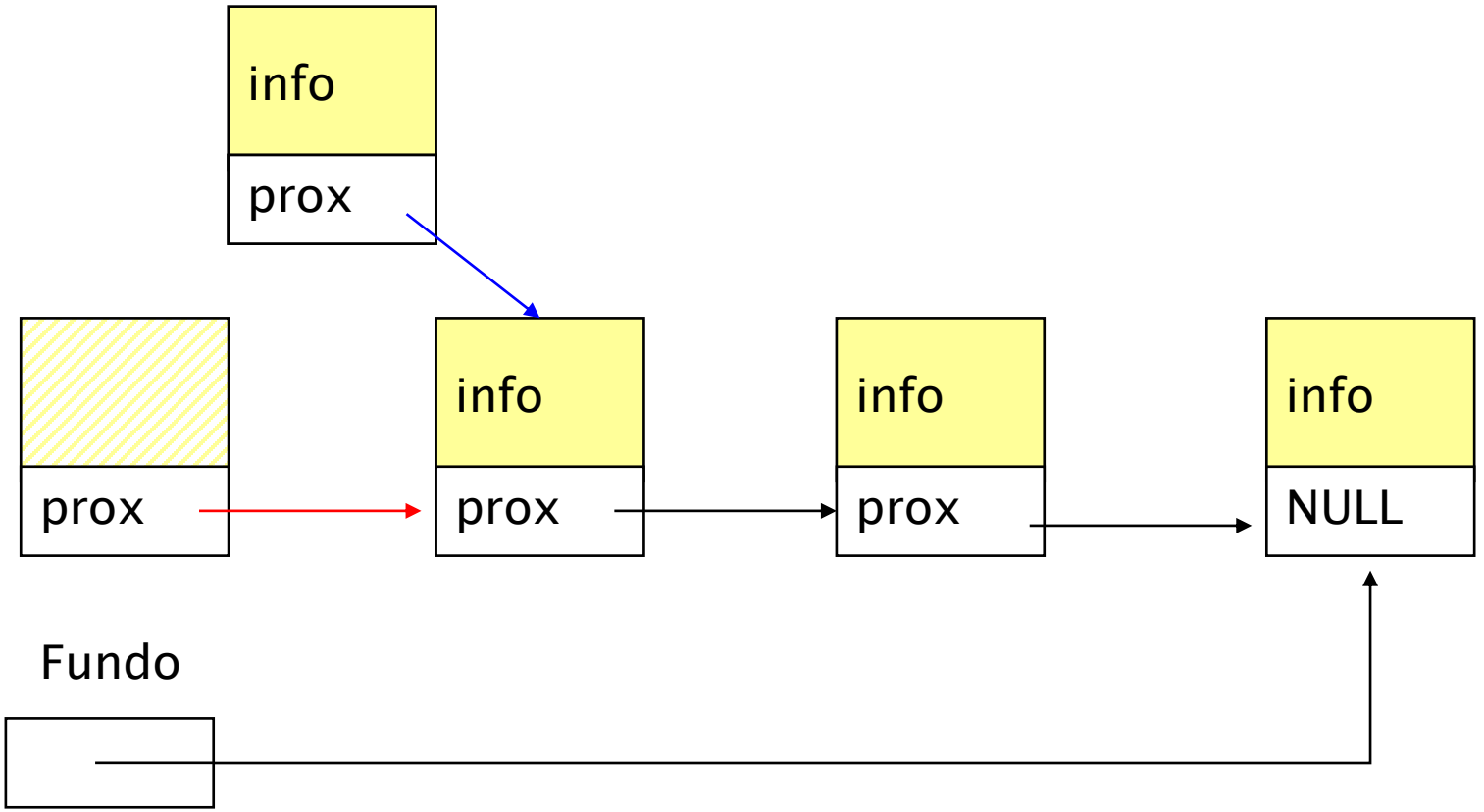
4. Implementação por Ponteiro

- **Inserir** na 1ª posição (1/3)



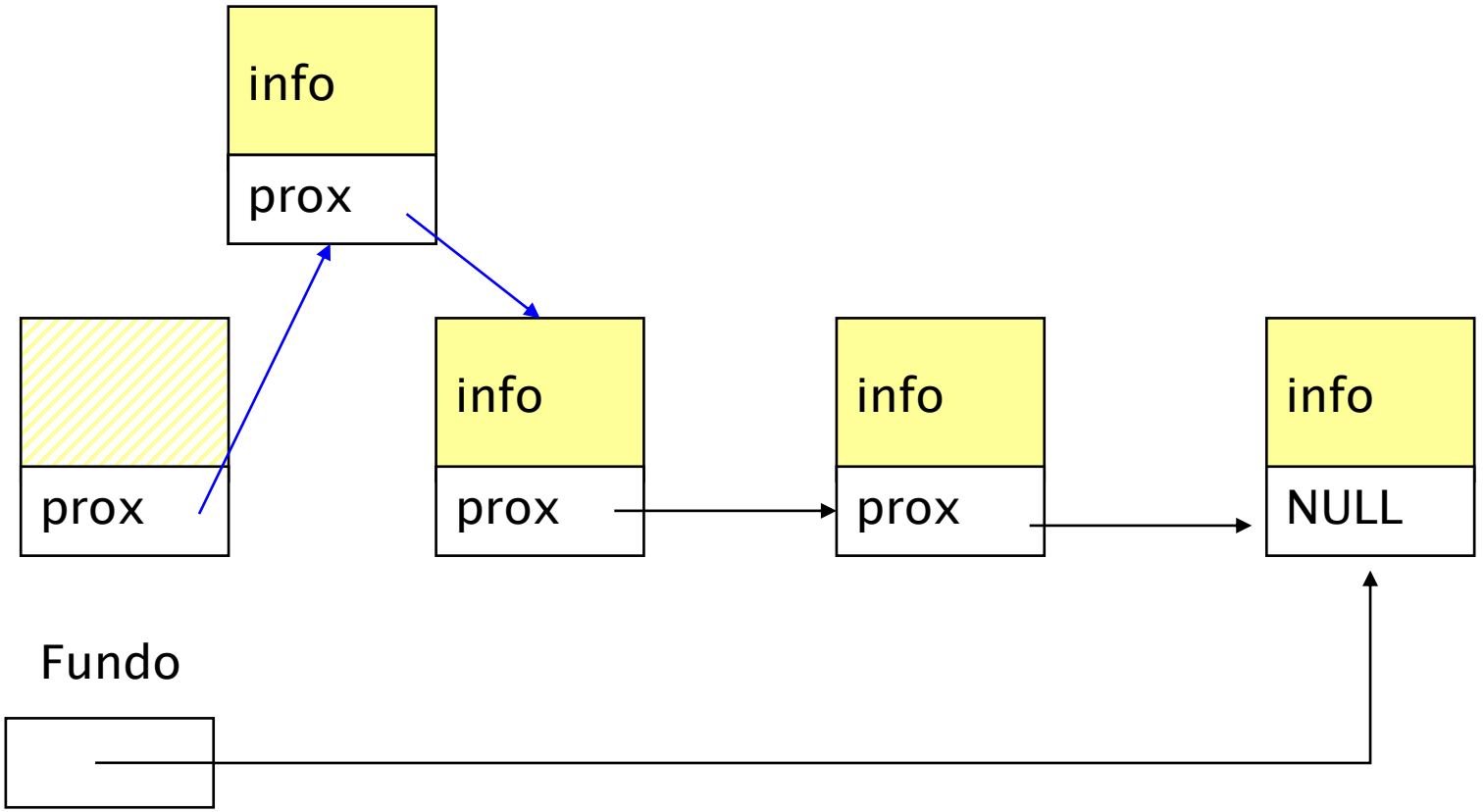
4. Implementação por Ponteiro

- **Inserir** na 1ª posição (2/3)



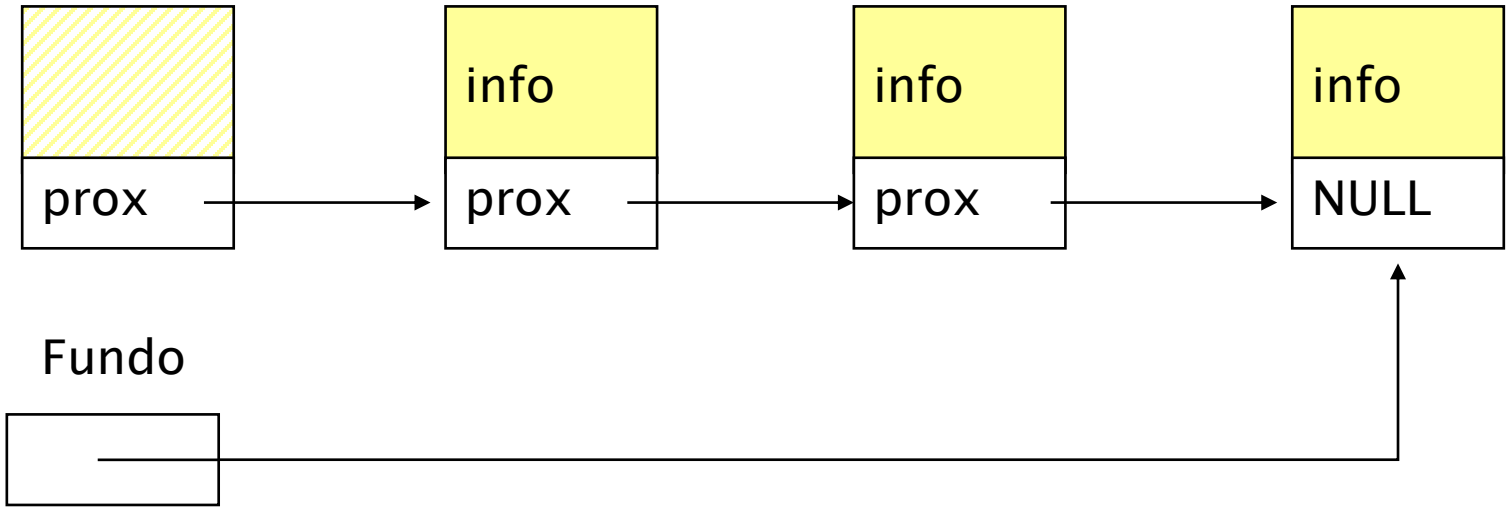
4. Implementação por Ponteiro

- **Inserir** na 1ª posição (3/3)



4. Implementação por Ponteiro

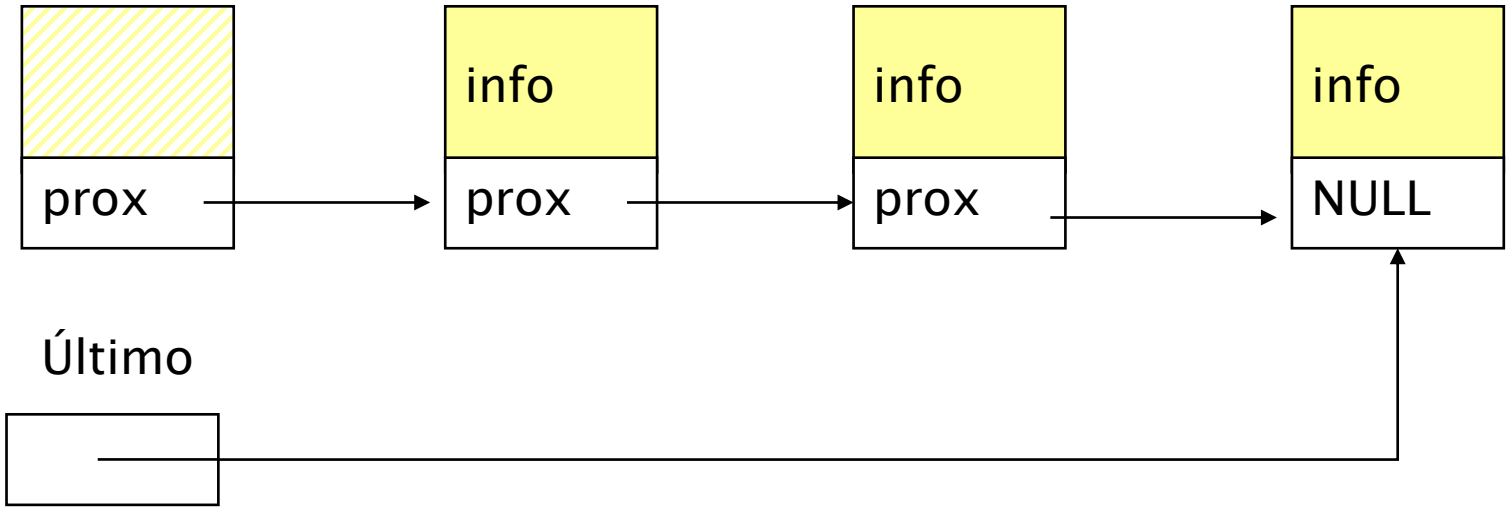
- Opção única de posição onde se pode **retirar**:



  - **Topo** da pilha, ou seja, **primeira** posição.

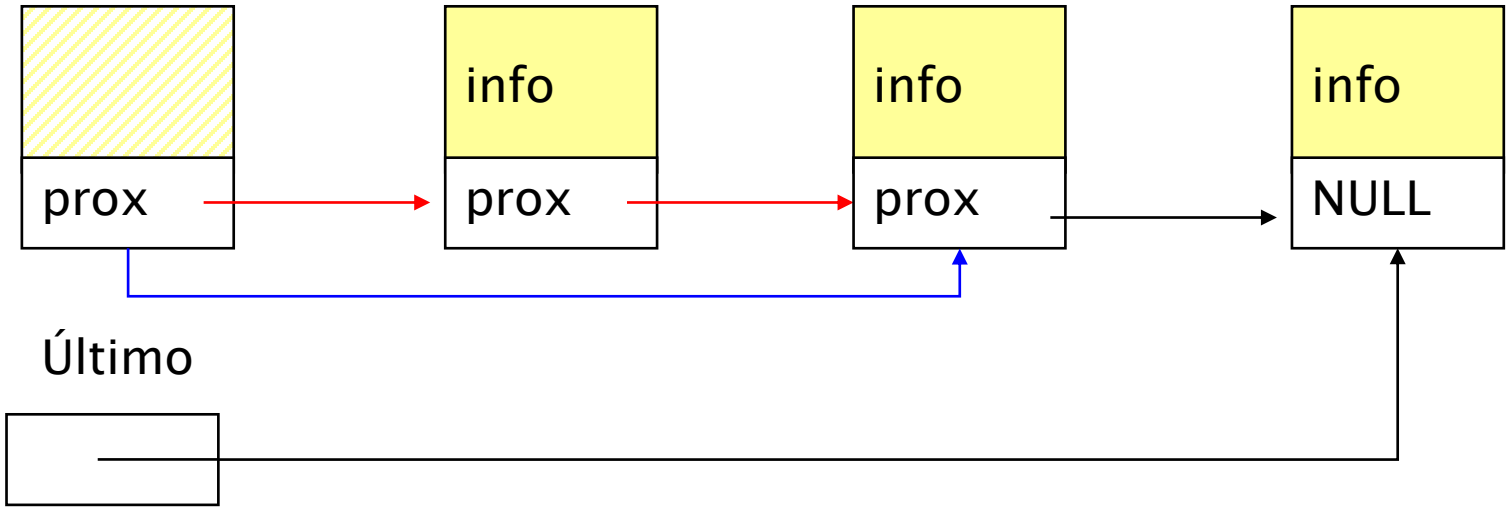
4. Implementação por Ponteiro

- **Retirar** da **1ª posição (1/3)**



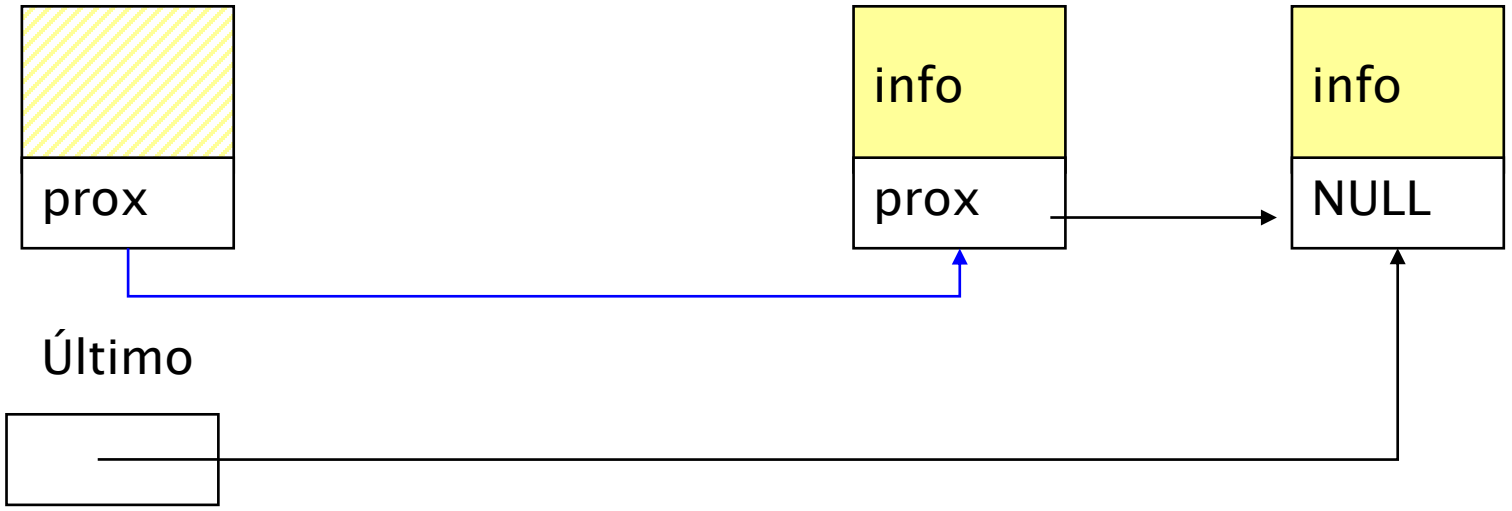
4. Implementação por Ponteiro

- **Retirar** da 1ª posição (2/3)



4. Implementação por Ponteiro

- **Retirar** da **1ª posição (3/3)**



4.1. Estrutura da Pilha

```
typedef int TChave;

typedef struct {
  TChave Chave;
  /* --- outros componentes --- */
}TItem;

typedef struct TCelulaEst {
  TItem item;
  struct TCelulaEst* pProx;
}TCelula;

typedef struct {
  TCelula* pFundo;
  TCelula* pTopo;
  int tamanho;
}TPilha;
```

4.2. Operações com Pilha

- **Com cabeça**

```
void FPVazia (TPilha* pPilha) {
  pPilha->pTopo = (TCelula*)malloc(sizeof(TCelula));
  pPilha->pFundo = pPilha->pTopo;
  pPilha->pTopo->pProx = NULL;
  pPilha->tamanho = 0;
}

int PVazia (TPilha* pPilha) {
  return (pPilha->pTopo == pPilha->pFundo);
}
```

4.2. Operações com Pilha

- **Sem cabeça**

```
void FPVazia (TPilha* pPilha) {
  pPilha->pTopo = NULL;
  pPilha->pFundo = NULL;
  pPilha->tamanho = 0;
}

int PVazia (TPilha* pPilha) {
  return (pPilha->pTopo == NULL);
}
```

4.2. Operações com Pilha

- **Com cabeça**

```
int Empilha (TPilha* pPilha, TItem* pItem) {

  TCelula* pNovo;
  pNovo = (TCelula*)malloc(sizeof(TCelula));
  pPilha->pTopo->item = *pItem;
  pNovo->pProx = pPilha->pTopo;
  pPilha->pTopo = pNovo;
  pPilha->tamanho++;

  return 1;
}
```

4.2. Operações com Pilha

- **Sem cabeça**

```
int Empilha (TPilha* pPilha, TItem* pItem) {

  TCelula* pNovo;
  pNovo = (TCelula*) malloc(sizeof(TCelula));
  pNovo ->item = *pItem;
  pNovo ->pProx = pPilha->pTopo;
  pPilha->pTopo = pNovo;
  pPilha->tamanho++;

  return 1;
}
```

4.2. Operações com Pilha

- Com cabeça

```
int Desempilha (TPilha* pPilha, TItem* pItem) {

  TCelula* pAux; /* celula a ser removida */
  if (PVazia(pPilha))
    return 0;

  pAux = pPilha->pTopo;
  pPilha->pTopo = pAux->pProx;
  *pItem = pAux->pProx->item;
  free(pAux);
  pPilha->tamanho--;

  return 1;
}
```

4.2. Operações com Pilha

- **Sem cabeça**

```
int Desempilha (TPilha* pPilha, TItem* pItem) {

  TCelula* pAux; /* celula a ser removida */
  if (PVazia(pPilha))
    return 0;

  pAux = pPilha->pTopo;
  pPilha->pTopo = pAux->pProx;
  *pItem = pAux->item;
  free(pAux);
  pPilha->tamanho--;
  return 1;
}
```

4.2. Operações com Pilha

- **Com cabeça e Sem cabeça**

```
int PTamanho (TPilha* pPilha) {
  return (pPilha->tamanho);
}
```

5. Exemplo: Editor de Textos

- Vamos escrever um Editor de Textos (ET) que aceite os comandos:
  - Cancela caractere
  - Cancela linha
  - Imprime linha

- O ET deverá ler um caractere de cada vez do texto de entrada e produzir a impressão linha a linha, cada linha contendo no máximo 70 caracteres de impressão.

- O ET deverá utilizar o tipo abstrato de dados Pilha definido anteriormente, implementado por meio de array.

5. Exemplo: Editor de Textos

- **#:** cancelar caractere anterior na linha sendo editada.
  - Ex.: **UFM#OB#P DCC##ECSI!**
- **\:** cancela todos os caracteres anteriores na linha sendo editada.
- **"*":** salta a linha.
- **"!":** Imprime os caracteres que pertencem à linha sendo editada, iniciando uma nova linha de impressão a partir do caractere imediatamente seguinte ao caractere salta-linha.
  - Ex.: **DECSI*UFOP*!**

    DECSI

    UFOP

5. Exemplo: Editor de Textos

- Sugestão de Texto para Testar o ET

```
Este et# um teste para o ET, o
extraterrestre em C.*Acabamos de
testar a capacidade de o ET saltar
de linha, utilizando seus poderes
extras (cuidado, pois agora vamos
estourar a capacidade máxima da
linha de impressão, que é de 70
caracteres.)*!O k#cut#rso dh#e
Estruturas de Dados et# h#um
cuu#rsh#o #x# x?*!#?!#+.* Como et#
bom n#nt#ao### r#ess#tt#ar
mb#aa#triz#cull#ado nn#x#ele!\
Sera que este funciona\\\? O
sinal? não### deve ficar! ~
```

5. Exemplo: Editor de Textos

- Implementação:
  - Este programa utiliza um tipo abstrato de dados sem conhecer detalhes de sua implementação.
  - A implementação do TAD Pilha que utiliza array pode ser substituída pela implementação que utiliza apontadores sem causar impacto no programa.

5. Exemplo: Editor de Textos

```
int main ( ) {
  TPilha Pilha;
  TItem x;
  FPVazia(&Pilha);
  x.Chave = getchar();
  while (x.Chave != '~') {
    if (x.Chave == '#') {
      if (!PVazia (&Pilha))
        Desempilha(&Pilha, &x); }
    else if (x.Chave == '\\') FPVazia(&Pilha);
    else if (x.Chave == '*') PImprime(&Pilha);
    else {
      if (PTamanho (Pilha) == MaxTam) PImprime(&Pilha);
        Empilha(&Pilha, &x); }
    x.Chave = getchar();
  }
  if (!PVazia (&Pilha)) PImprime(&Pilha);
  return 0;
}
```

5. Exemplo: Editor de Textos

```
void PImprime (TPilha* pPilha) {
  TPilha Pilhaux;
  TItem x;
  FPVazia(&Pilhaux);
  while (!PVazia(pPilha)) {
    Desempilha(pPilha, &x);
    Empilha(&Pilhaux,&x);
  }
  while (!PVazia (&Pilhaux)) {
    Desempilha(&Pilhaux, &x);
    putchar(x.Chave);
  }
  putchar('\n');
}
```

6. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.
- LANGSAM,Y.; AUGENSTEIN, M.J.; TENENBAUM, A.M. Data Structures using C and C++, 2a edição . Prentice Hall of India. 2007.
- CORMEM, T. H.; et al. Introduction to algorithms, 3a edição, The MIT Press.
- DROZDEK A. Estrutura de dados e algoritmos em C++,1a edição Cengage Learning.