ESTRUTURA DE DADOS - PILHA

Estrutura de Dados I – ED1

Listas lineares

- São estruturas que permitem representar um conjunto de dados de forma a preservar a relação de ordem linear (total) entre eles.
- É composta de nós, os quais podem conter um dado primitivo ou não.
- Define-se uma lista linear como sendo o conjunto de n > 0 nós x1, x2..., xn, organizados estruturalmente de forma a refletir as posições relativas dos mesmos.

Listas lineares

- Uma lista é uma sequência ordenada de elementos do mesmo tipo.
- Por exemplo, um conjunto de fichas de clientes de uma loja, organizadas pela ordem alfabética dos nomes dos clientes. Neste fichário é possível introduzir uma nova ficha ou retirar uma velha, alterar os dados de um cliente etc.

Listas Lineares

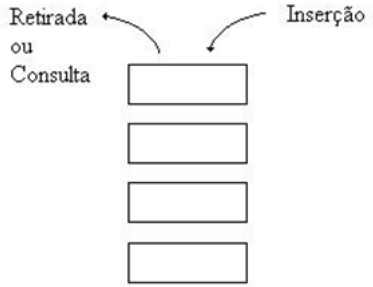
- Basicamente, o manuseio de listas lineares envolve três tipos de operações:
  1. Inserção de novos nodos à lista;
  2. Retirada de nodos da lista;
  3. Consulta de conteúdo de algum nodo da lista.

Listas lineares

- Existem outras operações que podem ser efetuadas sobre listas lineares:
  - Concatenar duas listas;
  - Determinar o número de nodos de uma lista;
  - Localizar um nodo da lista com um determinado conteúdo;
  - Criação de uma lista;
  - Cópia de uma lista;
  - Ordenação de uma lista por algum critério;
  - Destruição de uma lista;
  - Modificação do conteúdo de algum nodo da lista.

Pilha

- é uma lista linear em que todas as operações (inserção, retirada e consulta) são realizadas numa única extremidade da estrutura.
- Graficamente, uma pilha é representada por:



Pilha

- As extremidades de uma pilha são chamadas de BASE e TOPO

Pilha

- Uma vez que qualquer acesso é feito em uma única extremidade, o último elemento inserido numa pilha sempre será o primeiro a ser excluído, motivo pelo qual as pilhas são também chamadas de listas **LIFO** (do inglês *"Last-Input First-Output"*).

Pilha

- No âmbito computacional as pilhas são de grande utilidade, estando presentes, por exemplo, no controle de desvios e retornos de sub-programas, na análise da sintaxe de expressões aritméticas, na ordem de sobreposição de janelas abertas em aplicativos diversos, na implementação de recursividade, entre outros.

Implementaçã Estática

Pilha

- Para a implementação dos algoritmos que manipulam uma pilha com uso de memória estática, necessita-se de variáveis que fazem controle do tamanho da área onde a pilha será implementada, de onde estão as extremidades topo e base.
- *Obs: com a utilização de memória estática, a inclusão de um novo nodo na pilha é, na verdade, a ocupação de um nodo e não o acréscimo de um novo nodo na pilha. Da mesma forma, a retirada é a liberação de um nodo e não a exclusão de um nodo da pilha.*

Pilha

- Uma pilha vai usar uma estrutura composta de dados
- Um vetor do tipo de dados que irá abrigar
- Sobre este vetor se deverá criar uma estrutura lógica de controle para que o mesmo passe a funcionar como uma pilha
- Ou seja, somente incluir e retirar dados do topo da pilha e respeitar o tamanho da estrutura que abriga a base (início e fim da estrutura)

Pilha

- Para o controle do tamanho e das extremidades da pilha utiliza-se as seguintes variáveis:
- **tam** : armazena a capacidade da pilha
- **topo** : variável que possui o índice do nodo onde está o elemento mais recentemente inserido na pilha (o mais “novo” da pilha). Representa a extremidade topo da pilha.
- **base** : variável que possui o índice anterior ao índice do nodo onde está o elemento mais “antigo” na pilha. Representa a extremidade base da pilha.

Pilha

- Inicialmente, quando criada uma pilha, ela estará vazia, ou seja sem elementos inseridos.
- Portanto, os valores iniciais das variáveis de uma pilha são:

tam = tamanho da pilha

topo = -1

base = -1

Pilha

- Existem dois momentos especiais em uma pilha:
  - quando ela está vazia (nenhum elemento inserido)
  - e quando ela está cheia (todos os nodos ocupados)
  - Os testes lógicos que determinam quando uma pilha está em um dos momentos são:

Pilha vazia : **topo == base**

Pilha cheia : **topo == tam** -1

Pilha

Iniciando a estrutura lógica de uma pilha de inteiros:

int tam = 10;

- Tamanho da estrutura da pilha

int dados[tam];

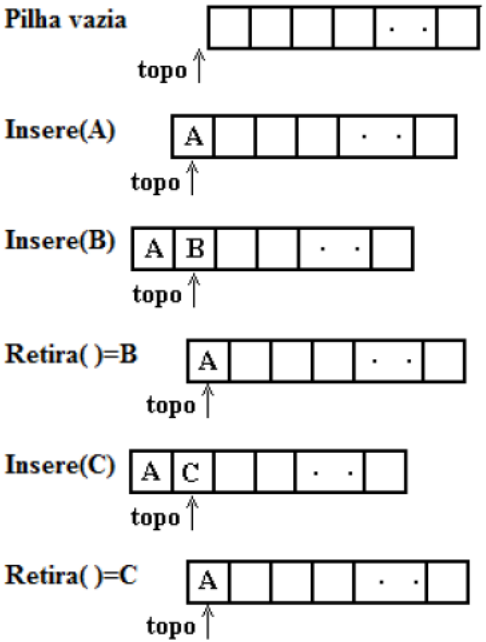
- Estrutura que armazenará a pilha na memória

int base = -1;

- O início da estrutura

int topo = base;

Inicio a pilha vazia topo apontando para o índice da base



Pilha - operações

- **push** (empurrar) - empilhar
  - insere um elemento na pilha
- **pop** (estourar/retirar) - desempilhar
  - Retira um elemento
- **peek** (espiar)
  - Lê o elemento no topo da pilha e não o retira

Pilha

1 push(H) - Coloca o H no topo

2 push(I) - Coloca o I no topo da Pilha

3 pop() - Retorna o elemento I

4 pop() - Retorna o elemento H

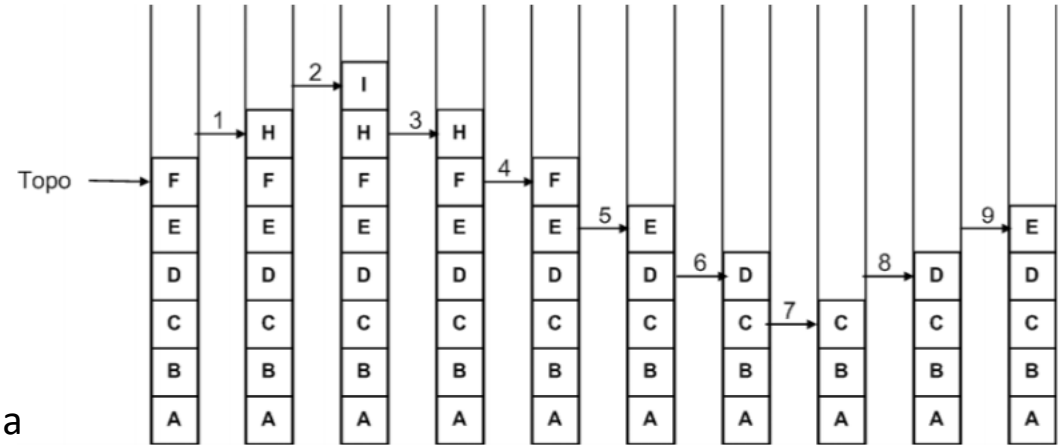
5 pop() - Retorna o elemento F

6 pop() - Retorna o elemento E

7 pop() - Retorna o elemento D

8 push(D) - Coloca o D no topo da Pilha

9 push(E) - Coloca o E no topo da Pilha



Pilha

- PRÉ-CONDIÇÃO - comuns
  - Se a pilha está vazia, cheia
- OPERAÇÃO
  - Só acontece se as pré-condições forem satisfeitas
  - Realizar a retirada, inserção ou leitura de elementos na pilha
- PÓS-OPERAÇÃO
  - Atualizar os apontadores lógicos da pilha
    - Topo

Pilha

- A pilha está vazia quando o topo estiver apontando para a base
  - if (topo == base)
    - Pilha vazia
      - Sendo a base = -1 , se o topo estiver em -1 significa que nada há na pilha
- A pilha está cheia quando o topo estiver apontando para o último elemento da pilha
  - if(topo == tam -1)
    - Pilha cheia
      - Se o topo estiver apontando para o último elemento da pilha

Pilha

- Diz-se que ocorre uma **situação de OVERFLOW** na pilha quando um novo elemento deve ser inserido mas não há mais nodos disponíveis, isto é, a **pilha está cheia**.
- O elemento não poderá ser inserido neste caso, ou seja, é uma tentativa de inclusão quando a pilha encontra-se cheia.

Pilha

- Ocorre uma **situação de UNDERFLOW** na pilha quando um elemento deve ser retirado da pilha e não há elementos na pilha, isto é, a **pilha está vazia**.
- O elemento não poderá ser retirado neste caso, ou seja, é uma tentativa de retirada quando a pilha encontra-se vazia.

Implementaçã Estática - Exemplo

Pilha

- Funções da estrutura pilha ficam em um arquivo **.hpp**
  - **.hpp** : são arquivos de cabeçalho que podem ser inseridos em arquivos .cpp
  - Podem ser usados em vários programas

```
1   #include <iostream>
2   using namespace std;
3
4   #include "pilha.hpp"
5
6   int main(void)
```

Pilha – pilha.hpp

Estrutura

```
struct Pilha
{
    int tam;
    int base;
    int topo;
    int *dados; //Vetor que será alocado para armazenar os elementos da pilha

    Pilha() //Construtor. Usado para inicializar os dados das variáveis da struct
    {
        tam=0;
        base = -1;
        topo = -1;
        dados = NULL;
    }
};
```

Pilha – pilha.hpp

Inicialização

```
///inicialização dos dados da pilha
void inicializaP(Pilha *p, int tam)
{
    p->base = -1; ///  (*p).base = -1;  /// p->base  p.topo
    p->topo = -1;
    p->tam  =  tam;
    p->dados = new int[tam];///aloca memória dinamicamente
}
```

Pilha – pilha.hpp

Verificar inicialização

```
//retorna true se a pilha foi inicializa
//retorna false se a pilha não foi inicializada
bool verificaInicializacaoP(Pilha *p) //verifica de a pilha foi inicializada
{
    if(p->dados != NULL)
        return true;
    else
        return false;
}
```

Pilha - pilha.hpp

Funções para verificar se cheia ou vazia

```
bool cheiaP(Pilha *p)
{
    if (p->topo == p->tam - 1)
        return true;
    else
        return false;
}

bool vaziaP(Pilha *p)
{
    if (p->topo == p->base)
        return true;
    else
        return false;
}
```

Pilha – pilha.hpp

Empilhar

```
///push
bool empilhaP(Pilha *p, int valor) ///push
{
    /// retorna false se a pilha não foi inicializada ou se ela está cheia
    if (verificaInicializacaoP(p)==false || cheiaP(p)==true)
        return false;
    else{
        p->topo++;
        p->dados[p->topo] = valor;
        return true;
    }
}
```

Pilha – pilha.hpp

Desempilhar

```
///pop
int desempilhaP(Pilha *p) ///pop
{

    int valor = 0; //inicializar a variável valor, a qual será retornada

    ///se a pilha foi inicializada && se não estiver vazia, retira valor
    if (vaziaP(p)==false)
    {
        valor = p->dados[p->topo];
        p->topo--;
    }

    return valor;
}
```

Pilha – pilha.hpp

Espiar

```
///peek
int espiaP(Pilha *p) ///peek
{
    int valor = 0; //inicializar a variável valor, a qual será retornada

    if (vaziaP(p)==false)
    {
        valor = p->dados[p->topo];
    }

    return valor;
}
```

Pilha – pilha.hpp

Mostrar

```
void mostraP(Pilha *p)
{

    cout << "PILHA: " << endl;
    cout << "TAM: " << p->tam << endl;
    cout << "TOPO: " << p->topo << endl;
    if(vaziaP(p) == false)
    {
        cout << "   ------------" << endl;
        for(int i = p->topo; i > p->base; i--){

            cout << setfill(' ') << std::setw(3) << i << "|";
            cout << setfill(' ') << std::setw(10) << p->dados[i] << "|" << endl;
            cout << "   ------------" << endl;
        }
    }
}
```

Pilha – pilha.hpp

Buscar

```
/// retorna true se o valor existe na pilha
/// retorna false se o valor não existe na pilha
bool buscaP(Pilha *p, int valor)
{

   for(int i = p->topo ; i > p->base; i--)
   {
       if (valor == p->dados[i])
          return true;
   }
   return false;

}
```

Pilha – pilha.hpp

Desalocar memória

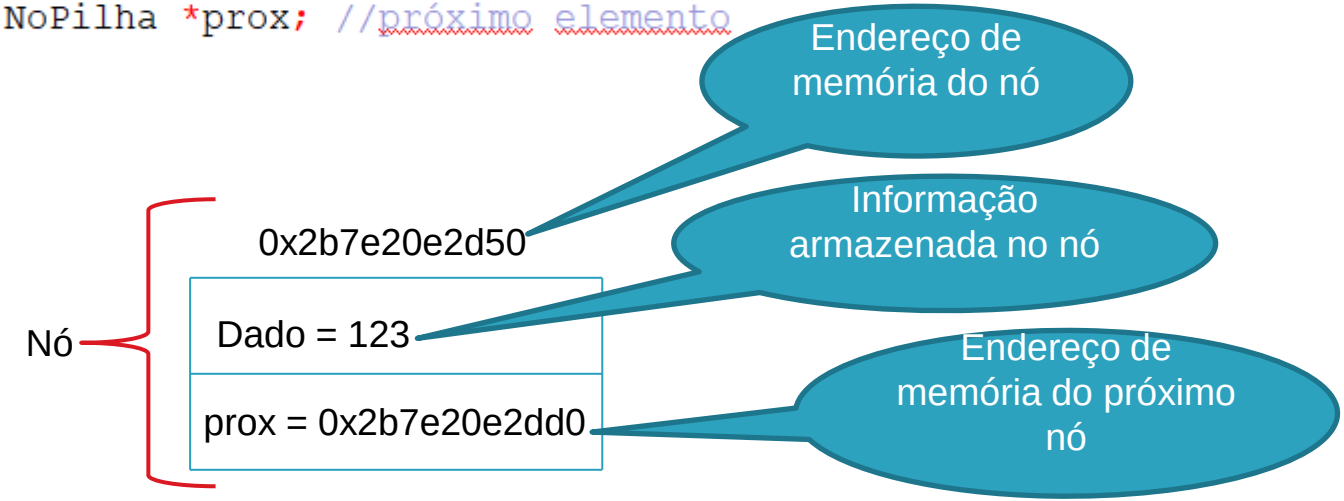
```
void destroiP(Pilha *p)
{
    p->base = -1;
    p->topo = -1;
    p->tam = 0;

    if(p->dados != NULL)
    {
        delete[] (p->dados);
        p->dados = NULL;
    }
}
```

Implementaçã dinâmica - Exemplo

Pilha dinâmica

Estrutura do nó/elemento

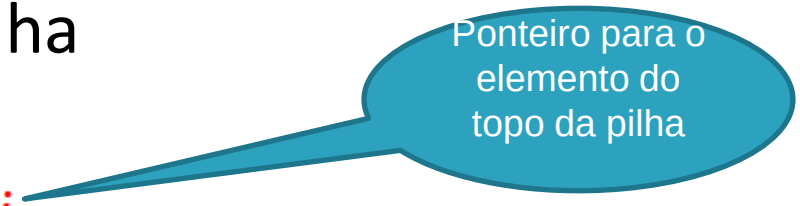
```
struct NoPilha
{
    int dado; //informação do nó
    NoPilha *prox; //próximo elemento
};
```



Pilha dinâmica

Estrutura da pilha

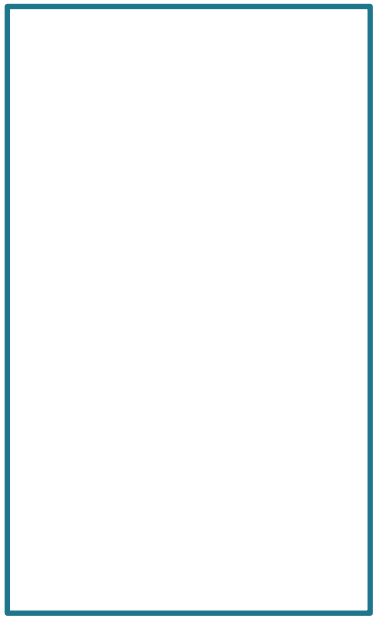
```
struct Pilha
{
    NoPilha *topo;

    Pilha(){ //Construtor. Inicialização da pilha
        topo = nullptr;
    }
};
```



Pilha dinâmica

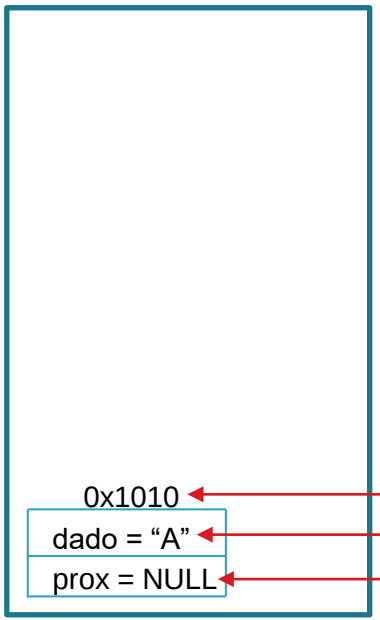
Pilha vazia

Topo = NULL



Pilha com um elemento

Topo = 0x1010 ← Endereço de memória do topo foi atualizado



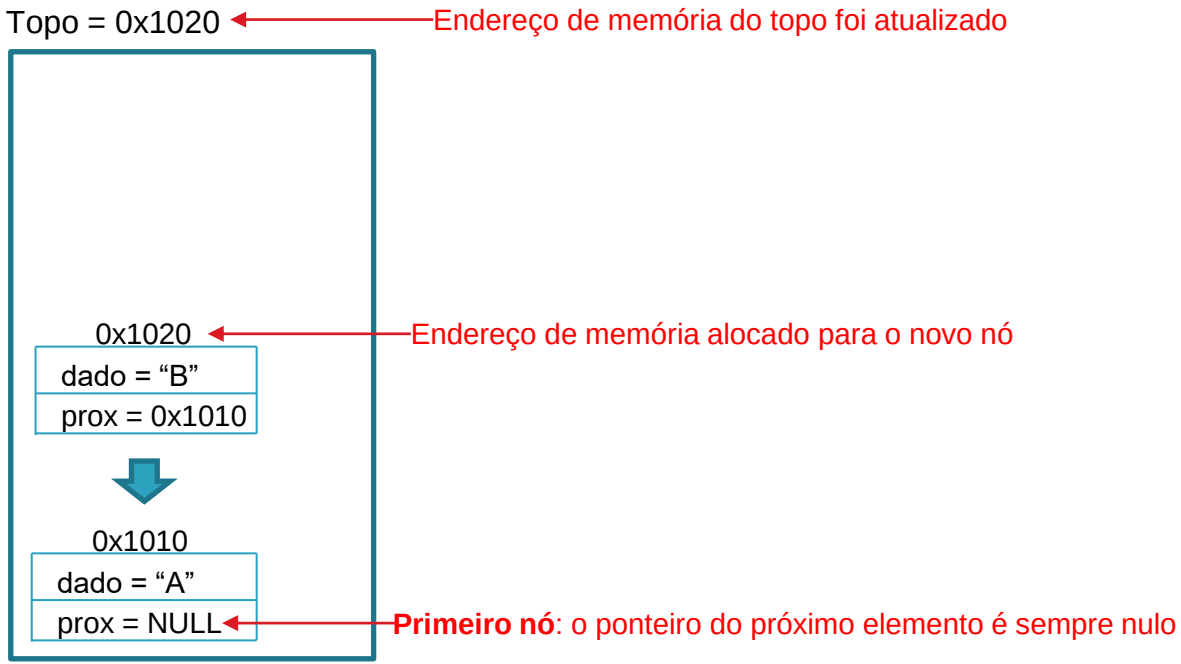
0x1010 ← Endereço de memória alocado para o novo nó

dado = “A” ← Dado

prox = NULL ← **Primeiro nó**: o ponteiro do próximo elemento é sempre nulo

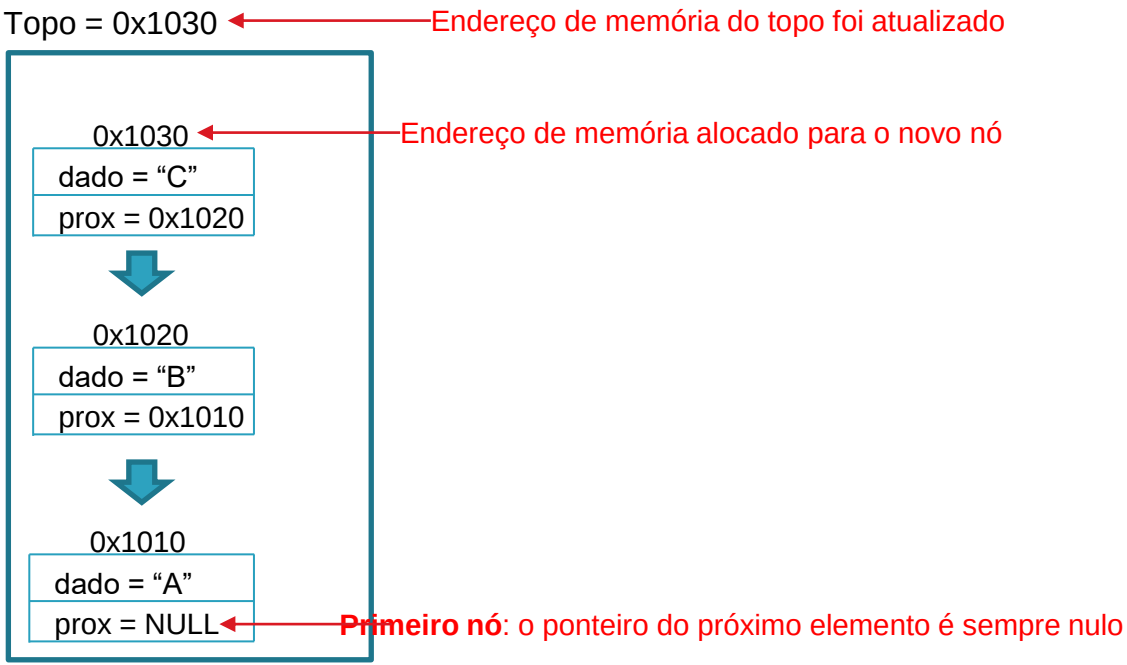
Pilha dinâmica

Empilhando novos elementos



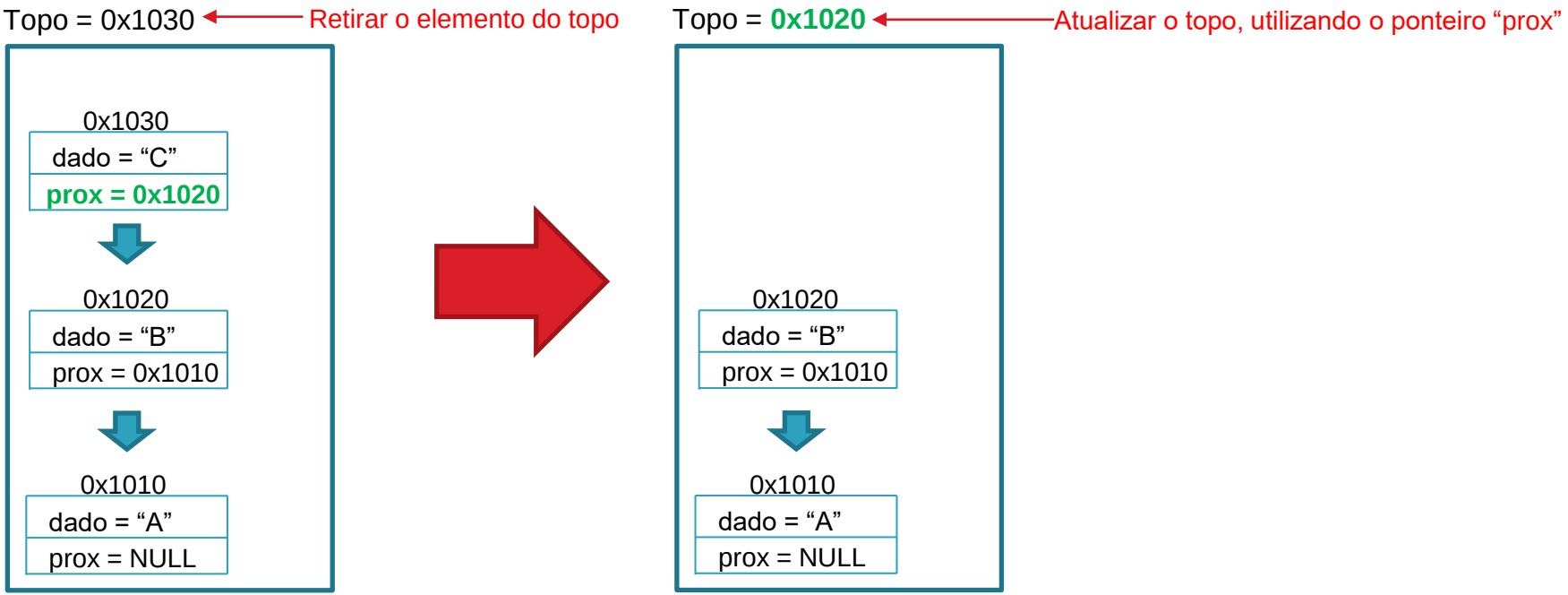
Pilha dinâmica

Empilhando novos elementos



Pilha dinâmica

Desempilhando elementos



Links

- https://www.cs.usfca.edu/~galles/visualization/StackArray.html
- https://visualgo.net/en/list