UFRJ

INSTITUTO DE COMPUTAÇÃO UFRJ

Programação de Computadores II

Pilha

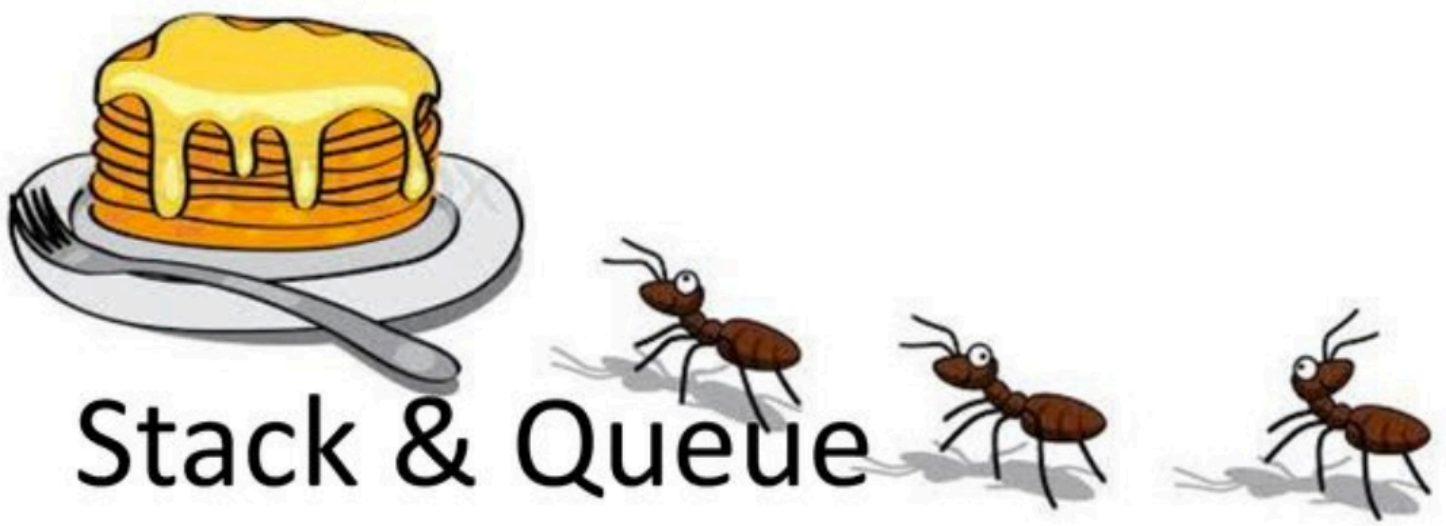
Profa. Giseli Rabello Lopes

Sumário

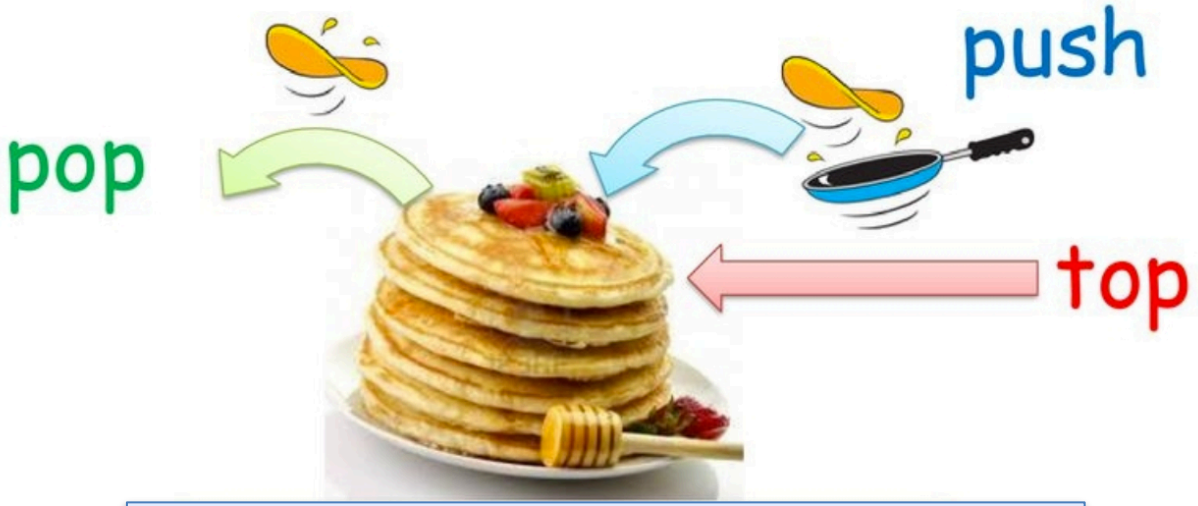
- Pilha
- Funções de gerenciamento
- Implementação estática
- Implementação dinâmica

Adaptado de material preparado por Profs. Luciano Digiampietri e Norton T. Roman
e material de apoio do livro Data Structures and Algorithm Analysis in C, C++ by Mark Allen Weiss

Pilha e Fila



Pilha



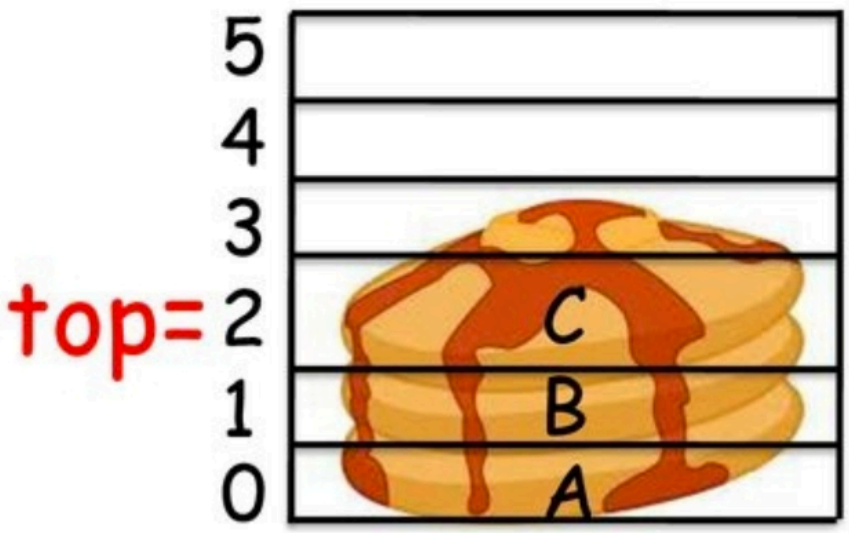
LIFO (Last In First Out)

- Uma coleção ordenada de itens (estrutura linear)
- O final é chamado de topo da pilha
- Itens são inseridos no topo da pilha —> **push**
- Itens são removidos do topo da pilha —> **pop**

Funções de gerenciamento

- Inicializar a estrutura
- Retornar a quantidade de elementos válidos
- Exibir os elementos da estrutura
- Inserir elementos na estrutura (push)
- Excluir elementos da estrutura (pop)
- Reinicializar a estrutura
- Verificar se a pilha está vazia (apenas para versão dinâmica)

Pilha - Implementação estática



Pilha - Implementação estática

- Utilizaremos um arranjo (*array*) de elementos de tamanho predefinido
- Controlaremos a posição do elemento que está no topo da pilha

Modelagem (pilha estática)

```
#include <stdio.h>
#define MAX 50

#define true 1
#define false 0

typedef int bool;

typedef int TIPOCHAVE;
```

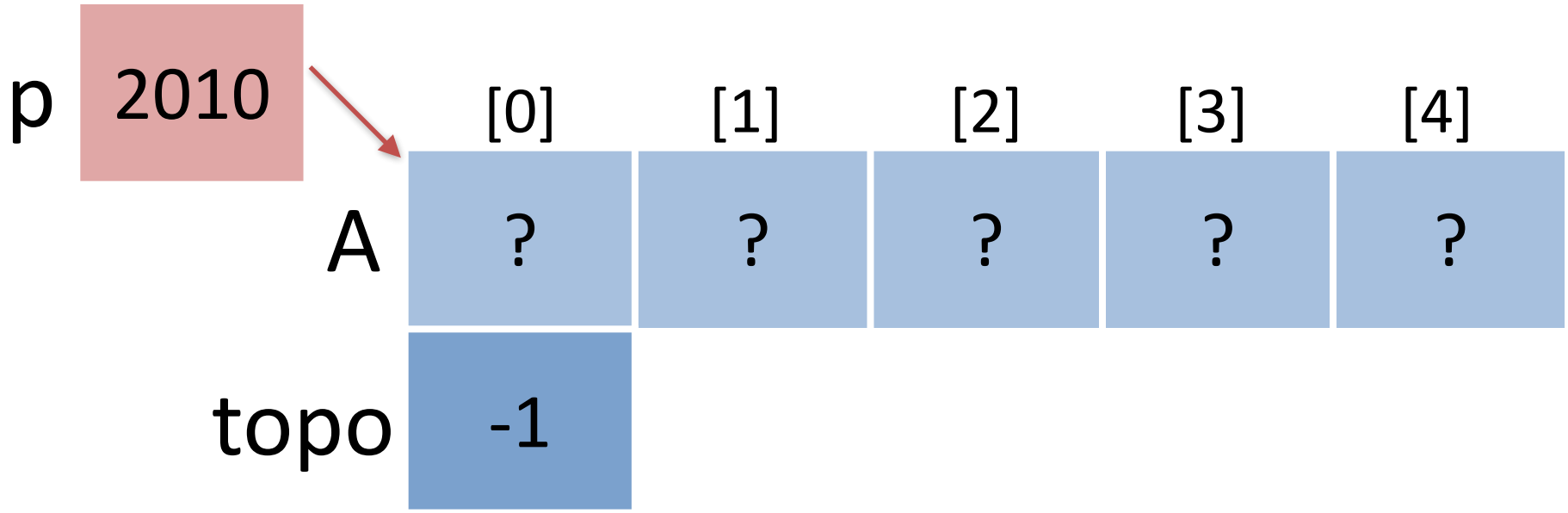
```
typedef struct {
  TIPOCHAVE chave;
} REGISTRO;

typedef struct {
  REGISTRO A[MAX];
  int topo;
} PILHA;
```

Inicialização

- Para inicializar uma pilha já criada pelo usuário, precisamos apenas acertar o valor do campo topo
- Já que o topo indicará a posição no arranjo do elemento que está no topo da pilha e a pilha está vazia, iniciaremos esse campo com valor -1

Inicialização

```
void inicializarPilha(PILHA* p)
{
  p->topo = -1;
}
```

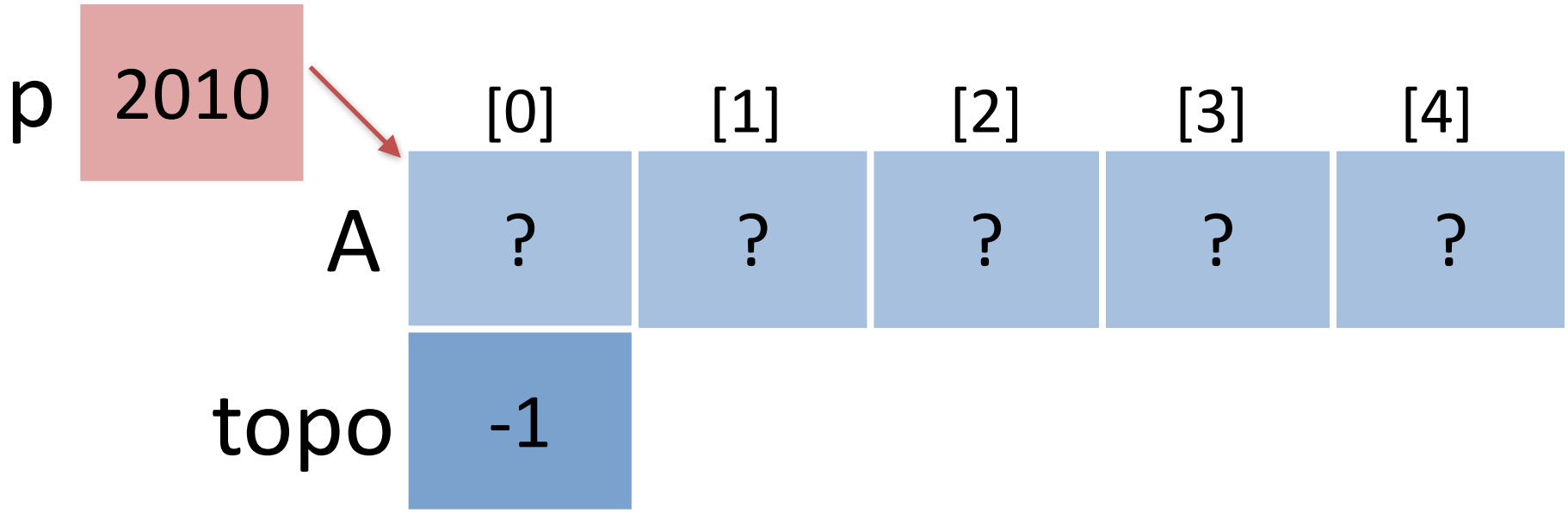


Retornar número de elementos

- Já que o campo topo contém a posição no arranjo do elemento no topo da pilha, o número de elementos é igual a: topo + 1
- Notem que para a pilha vazia isto também funciona

Retornar número de elementos

```
int tamanhoPilha(PILHA* p)
{
   return p->topo + 1;
}
```

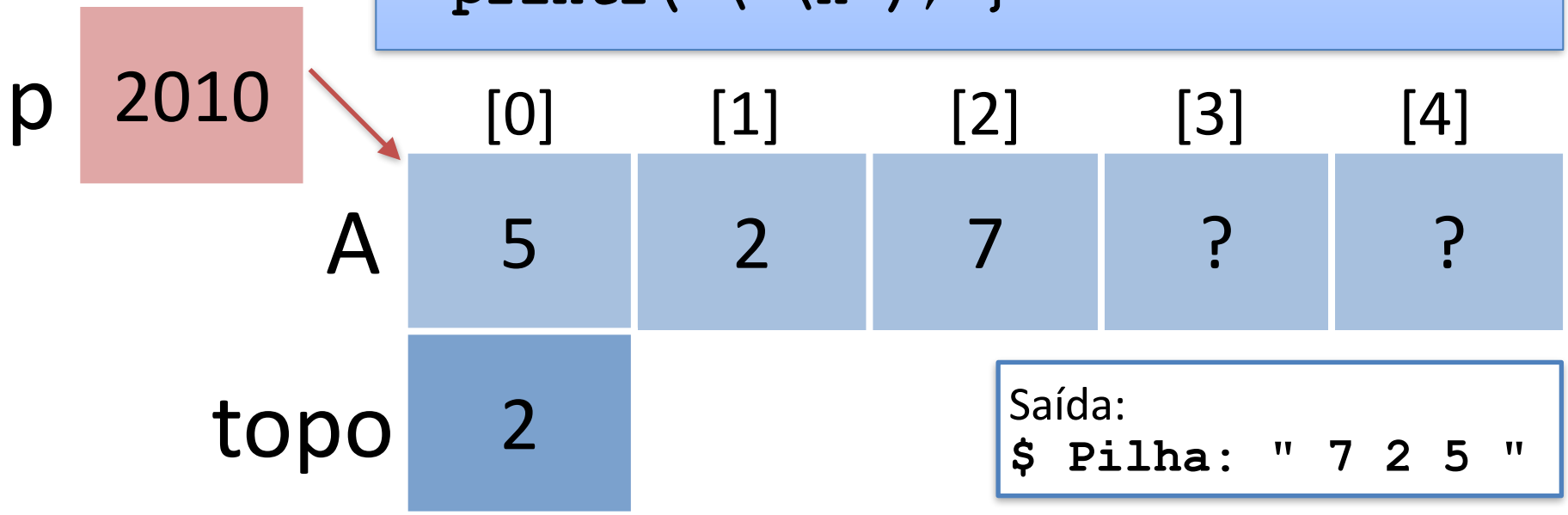


Exibição/Impressão

- Para exibir os elementos da estrutura precisaremos iterar pelos elementos válidos e, por exemplo, imprimir suas chaves

Exibição/Impressão

```
void exibirPilha(PILHA* p) {
  printf("Pilha: \" ");
  int i;
  for (i=p->topo;i>=0;i--) {
    printf("%i ", p->A[i].chave); }
  printf("\"\n"); }
```



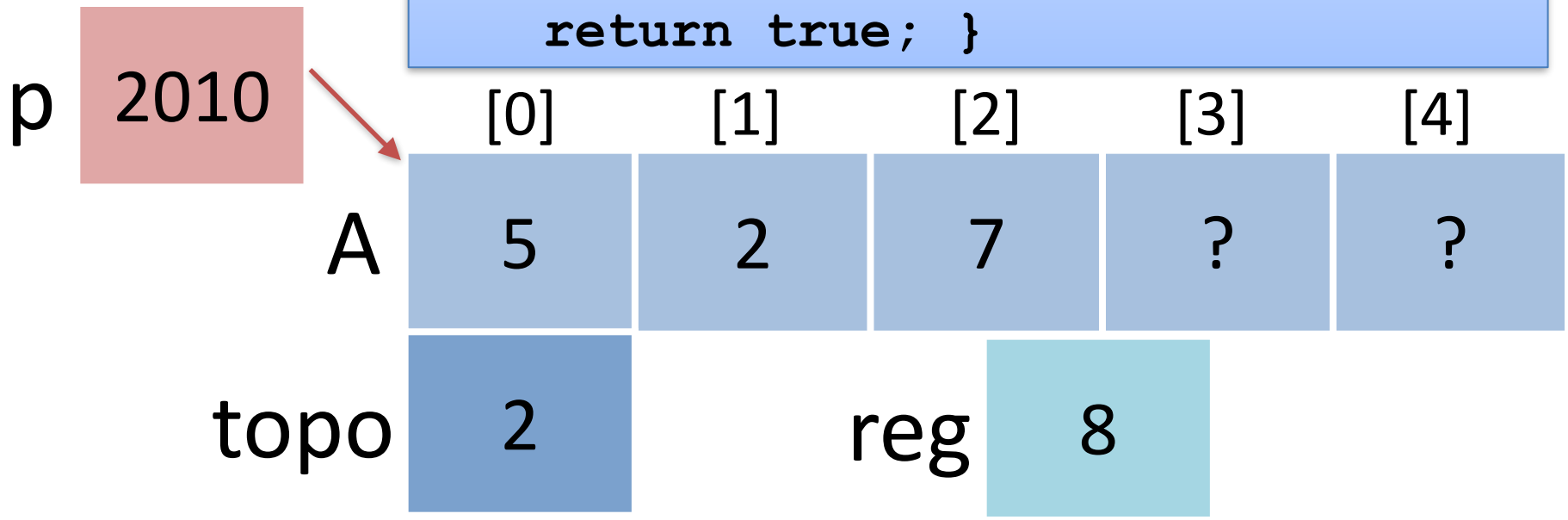
Saída:
`$ Pilha: " 7 2 5 "`

Inserção de um elemento (**`push`**)

- O usuário passa como parâmetro um registro a ser inserido na pilha
- Se a pilha não estiver cheia, o elemento será inserido no topo da pilha, ou melhor, "acima" do elemento que está no topo da pilha

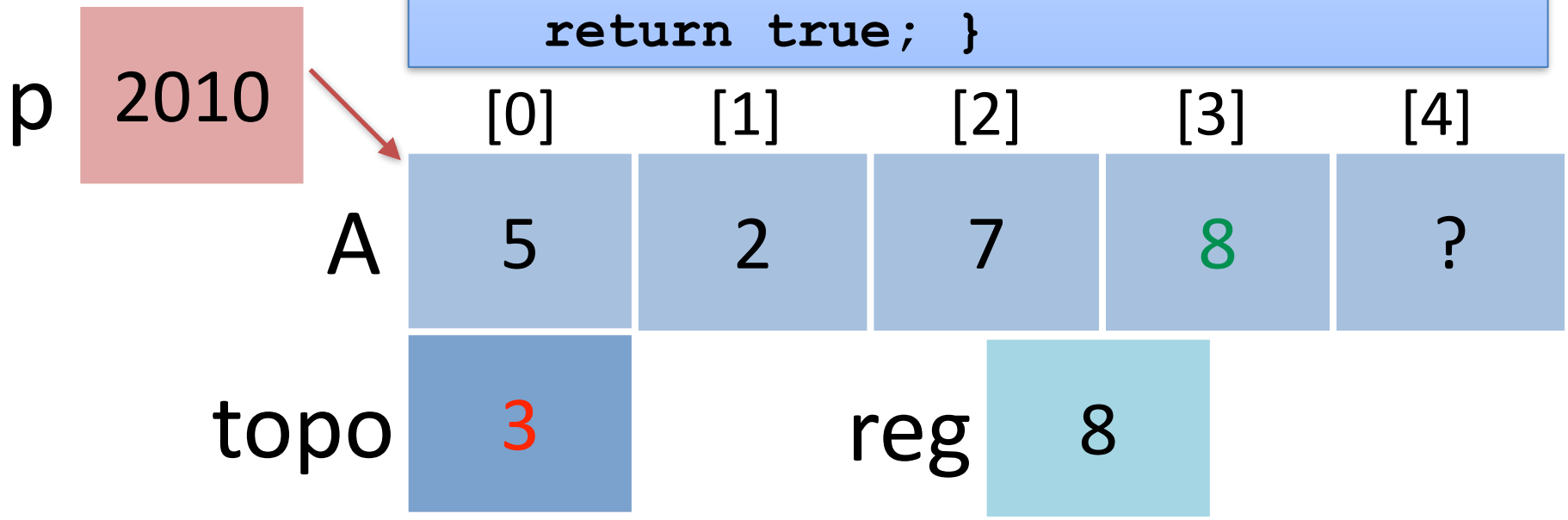
Inserção de um elemento (**push**)

```
bool inserirElementoPilha(PILHA* p,
REGISTRO reg) {
    if (p->topo >= MAX-1)
       return false;
    p->topo = p->topo+1;
    p->A[p->topo] = reg;
    return true; }
```



Inserção de um elemento (**push**)

```
bool inserirElementoPilha(PILHA* p,
REGISTRO reg) {
    if (p->topo >= MAX-1)
       return false;
    p->topo = p->topo+1;
    p->A[p->topo] = reg;
    return true; }
```

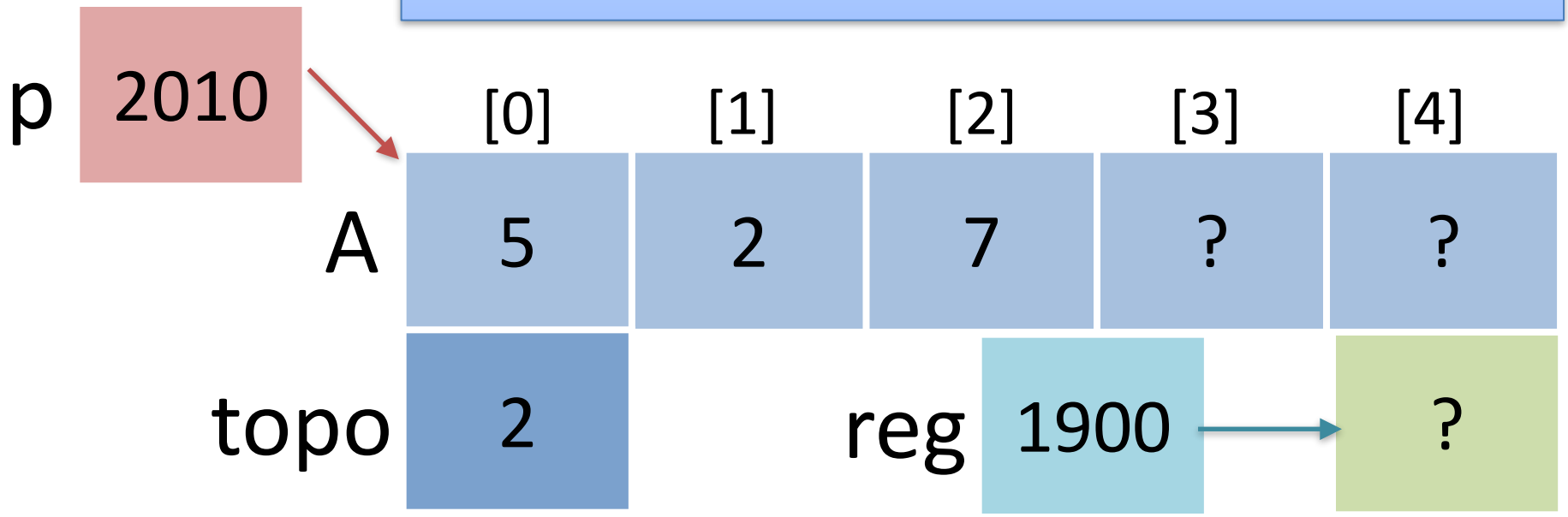


Exclusão de um elemento (pop)

- O usuário solicita a exclusão do elemento do topo da pilha:
  - Se a pilha não estiver vazia, além de excluir esse elemento da pilha iremos copiá-lo para um local indicado pelo usuário

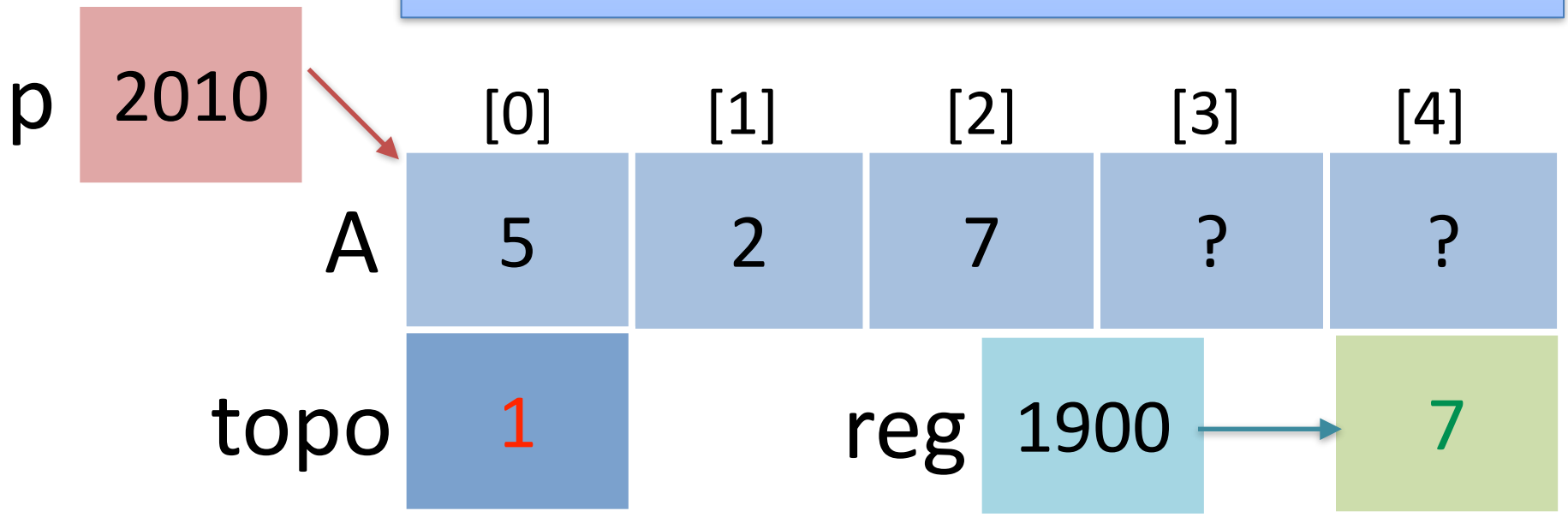
Exclusão de um elemento (pop)

```
bool excluirElementoPilha(PILHA* p,
REGISTRO* reg) {
    if (p->topo == -1) return false;
       *reg = p->A[p->topo];
    p->topo = p->topo-1;
    return true; }
```



Exclusão de um elemento (pop)

```
bool excluirElementoPilha(PILHA* p,
REGISTRO* reg) {
    if (p->topo == -1) return false;
   *reg = p->A[p->topo];
    p->topo = p->topo-1;
    return true; }
```

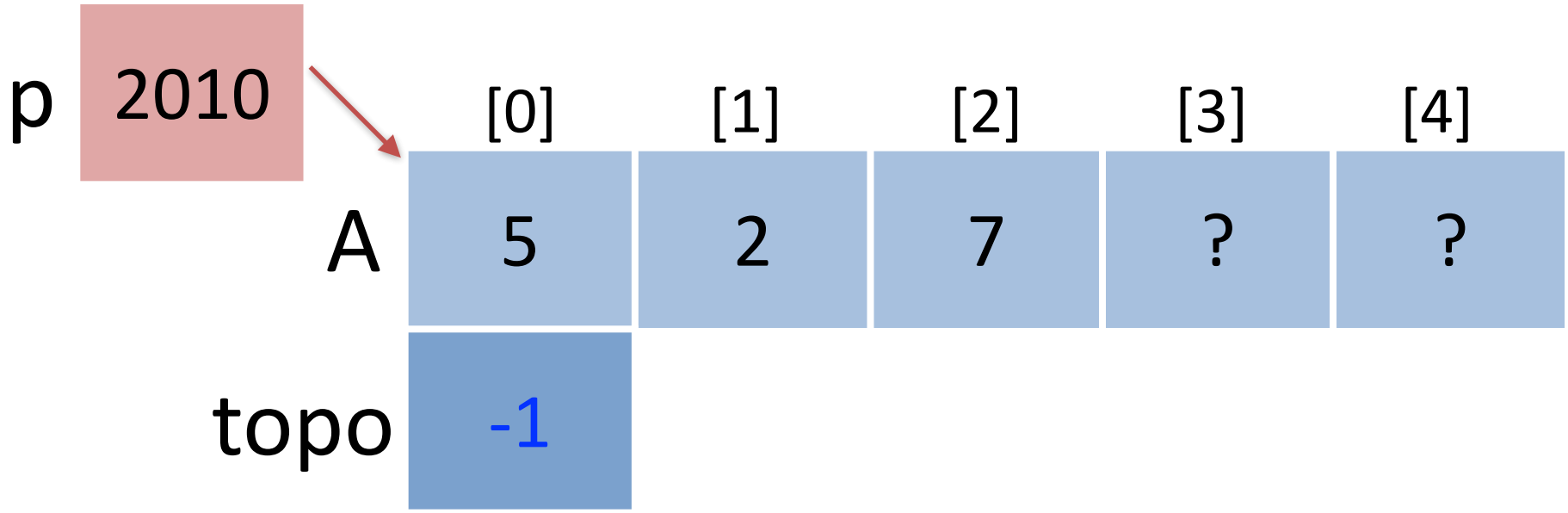


Reinicialização

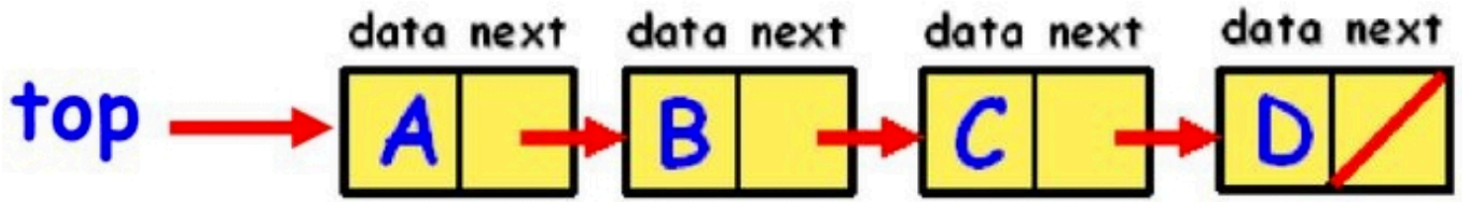
- Para esta estrutura, para reinicializar a pilha basta colocar -1 no campo topo

Reinicialização

```
void reinicializarPilha(PILHA* p) {
  p->topo = -1;
}
```



Pilha - Implementação dinâmica



Pilha - Implementação dinâmica

- Alocaremos e desalocaremos a memória para os elementos sob demanda
- **Vantagem:** não precisamos gastar memória que não estamos usando
- Cada elemento indicará quem é seu sucessor (quem está “abaixo” dele na pilha)
- Controlaremos o endereço do elemento que está no topo da pilha

Modelagem (pilha dinâmica)

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;

typedef int TIPOCHAVE;

typedef struct {
  TIPOCHAVE chave;
  //outros campos...
} REGISTRO;
```

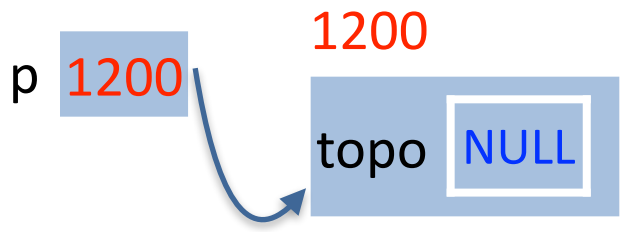
```
typedef struct aux {
  REGISTRO reg;
  struct aux* prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
  PONT topo;
} PILHA;
```

Inicialização

- Para inicializar uma pilha já criada pelo usuário, precisamos apenas acertar o valor do campo topo
- Já que o topo conterá o endereço do elemento que está no topo da pilha e a pilha está vazia, iniciaremos esse campo com valor ***NULL***

Inicialização

```
void inicializarPilha(PILHA* p)
{
   p->topo = NULL;
}
```

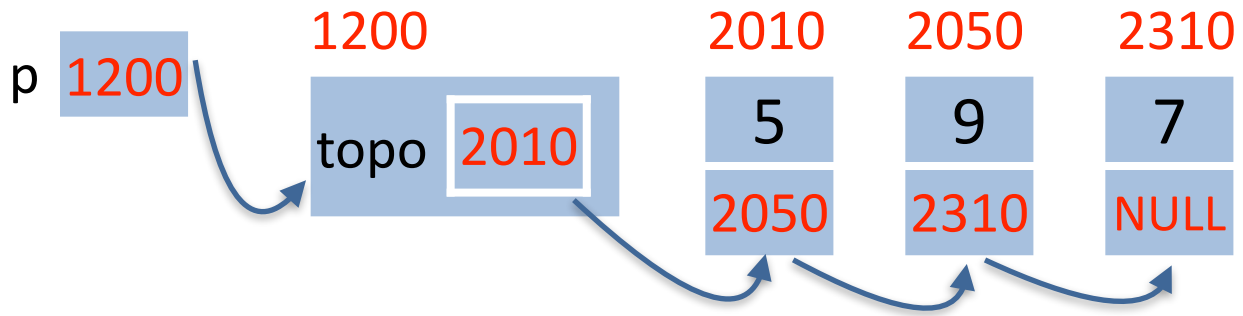


Retornar número de elementos

- Já que não temos um campo com o número de elementos na pilha, precisaremos percorrer todos os elementos para contar quantos são

Retornar número de elementos

```
int tamanho(PILHA* p) {
   PONT end = p->topo;
   int tam = 0;
   while (end != NULL) {
      tam++;
      end = end->prox; }
   return tam; }
```



Verificar se a pilha está vazia

- Por que não usar a função **`tamanho`** para verificar se a pilha está vazia?
- É bem mais simples verificar se topo está armazenando o endereço **`NULL`**

Verificar se a pilha está vazia

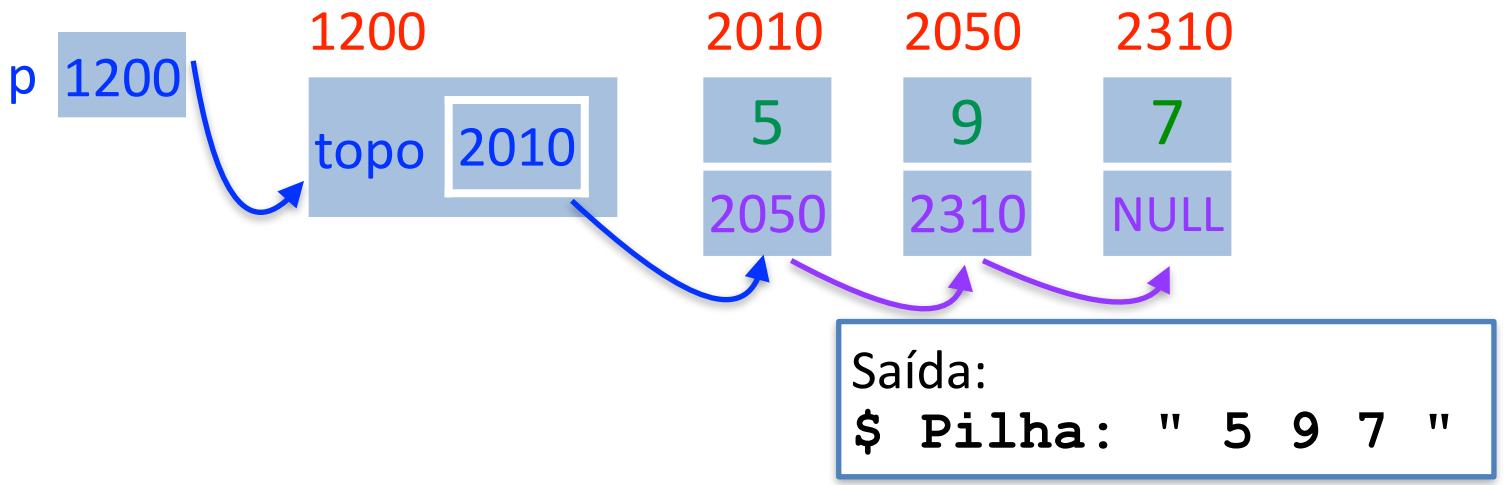
```
bool estaVazia(PILHA* p) {
   if (p->topo == NULL)
      return true;
   return false;
}
```

Exibição/Impressão

- Para exibir os elementos da estrutura precisaremos percorrer os elementos (iniciando pelo elemento do topo da pilha) e, por exemplo, imprimir suas chaves

Verificar Exibição/Impressão

```
void exibirPilha(PILHA* p) {
  PONT end = p->topo;
  printf("Pilha: \" ");
  while (end != NULL) {
    printf("%i ", end->reg.chave);
    end = end->prox; }
  printf("\"\n"); }
```

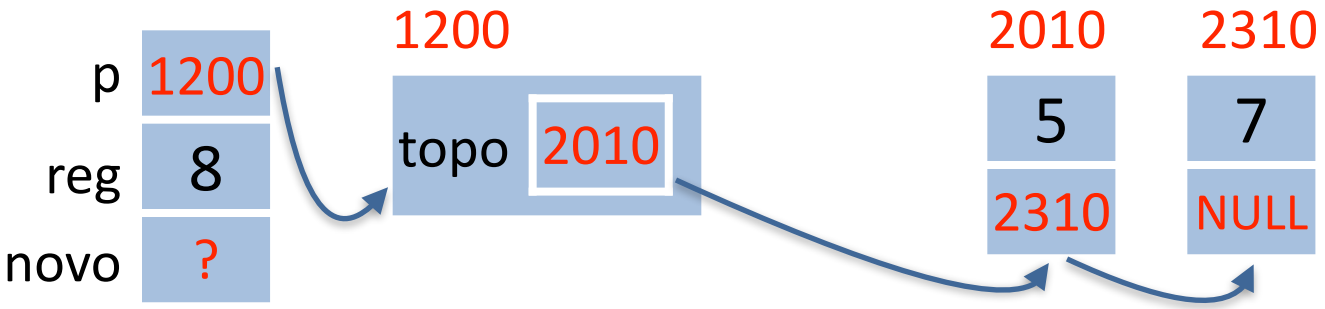


Inserção de um elemento (**`push`**)

- O usuário passa como parâmetro um registro a ser inserido na pilha
- O elemento será inserido no topo da pilha, ou melhor, "acima" do elemento que está no topo da pilha
  O novo elemento irá apontar para o elemento que estava no topo da pilha

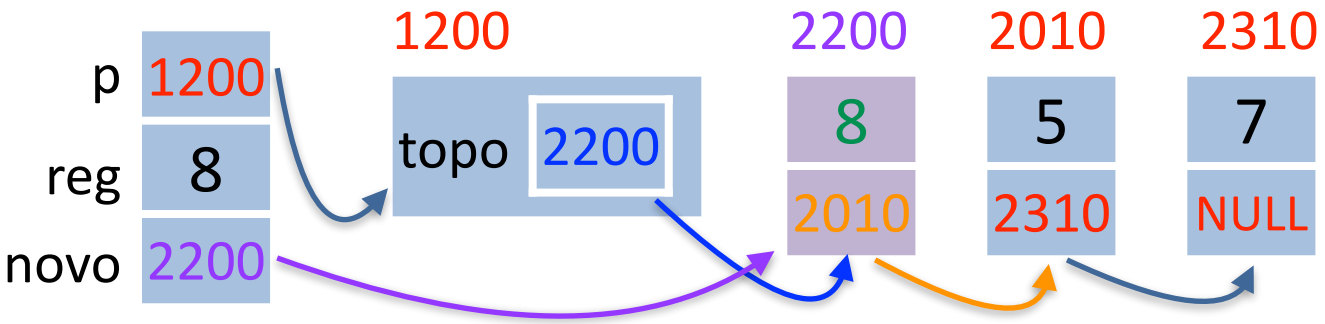
Inserção de um elemento (**push**)

```
bool inserirElemPilha(PILHA* p, REGISTRO reg) {
   PONT novo = (PONT) malloc(sizeof(ELEMENTO));
   novo->reg = reg;
   novo->prox = p->topo;
   p->topo = novo;
   return true;
}
```



Inserção de um elemento (**push**)

```
bool inserirElemPilha(PILHA* p, REGISTRO reg) {
   PONT novo = (PONT) malloc(sizeof(ELEMENTO));
   novo->reg = reg;
   novo->prox = p->topo;
   p->topo = novo;
   return true;
}
```



Exclusão de um elemento (pop)

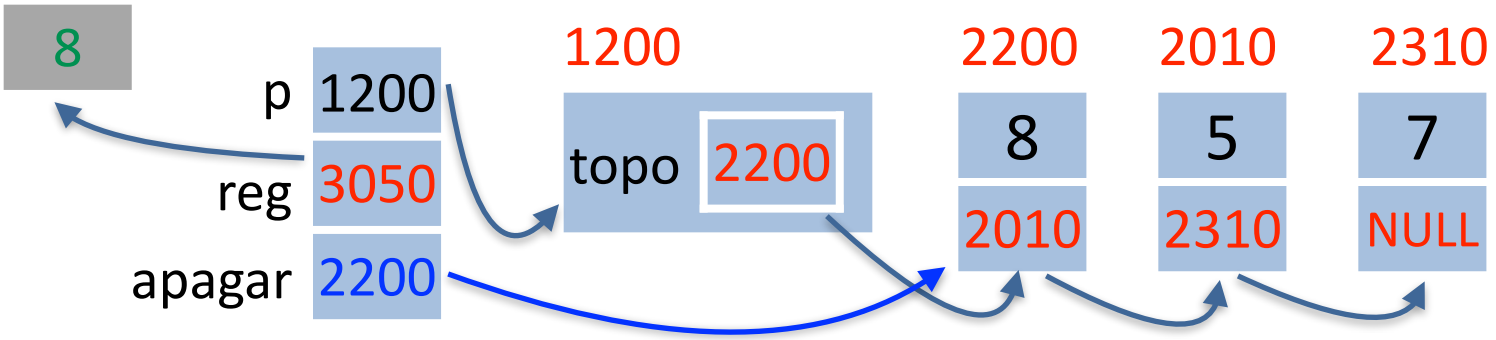
- O usuário solicita a exclusão do elemento do topo da pilha:
  - Se a pilha não estiver vazia, além de excluir esse elemento da pilha iremos copiá-lo para um local indicado pelo usuário

Exclusão de um elemento (pop)

```
bool excluirElemPilha(PILHA* p, REGISTRO* reg) {
  if ( p->topo == NULL) return false;
  *reg = p->topo->reg;
  PONT apagar = p->topo;
  p->topo = p->topo->prox;
  free(apagar);
  return true; }
```

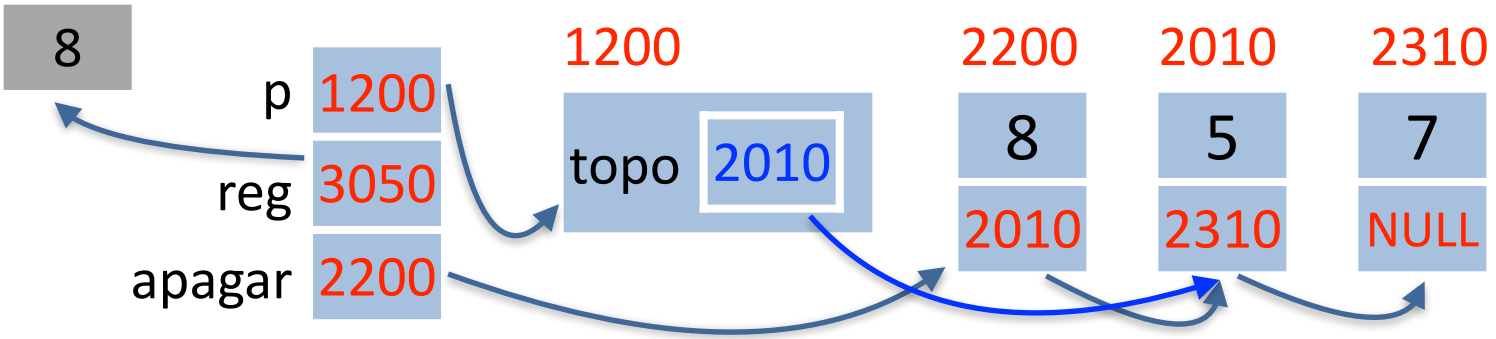
Exclusão de um elemento (pop)

```
bool excluirElemPilha(PILHA* p, REGISTRO* reg) {
  if ( p->topo == NULL) return false;
  *reg = p->topo->reg;
  PONT apagar = p->topo;
  p->topo = p->topo->prox;
  free(apagar);
  return true; }
```



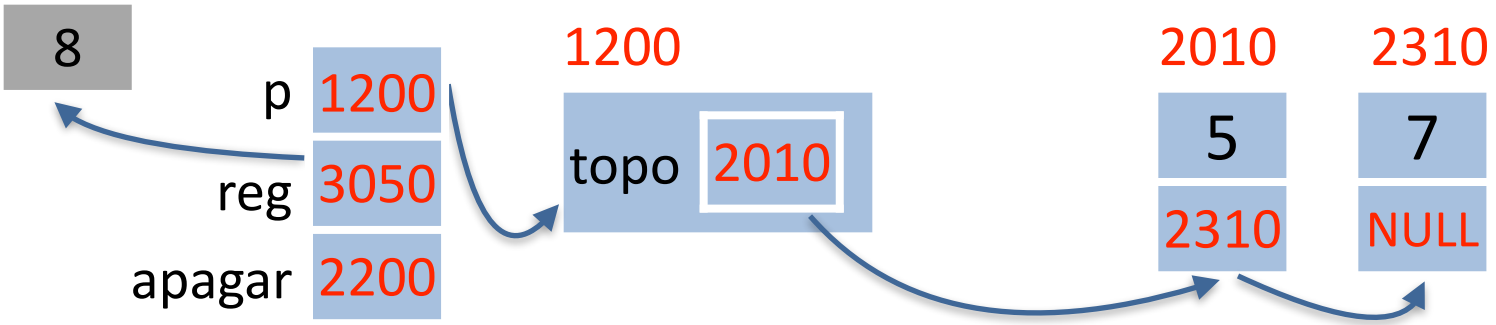
Exclusão de um elemento (pop)

```
bool excluirElemPilha(PILHA* p, REGISTRO* reg) {
   if ( p->topo == NULL) return false;
   *reg = p->topo->reg;
   PONT apagar = p->topo;
   p->topo = p->topo->prox;
   free(apagar);
   return true; }
```



Exclusão de um elemento (pop)

```
bool excluirElemPilha(PILHA* p, REGISTRO* reg) {
  if ( p->topo == NULL) return false;
  *reg = p->topo->reg;
  PONT apagar = p->topo;
  p->topo = p->topo->prox;
  free(apagar);
  return true; }
```



Reinicialização

- Para reinicializar a pilha, precisamos excluir todos os seus elementos e colocar **`NULL`** no campo **`topo`**

Reinicialização

```
void reinicializarPilha(PILHA* p) {
  PONT apagar;
  PONT posicao = p->topo;
  while (posicao != NULL) {
        apagar = posicao;
        posicao = posicao->prox;
        free(apagar); }
  p->topo = NULL; }
```