Estrutura de Dados

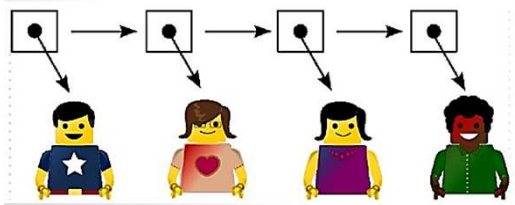
Aula 06 - Pilhas

Profa. Dra. Lúcia Guimarães

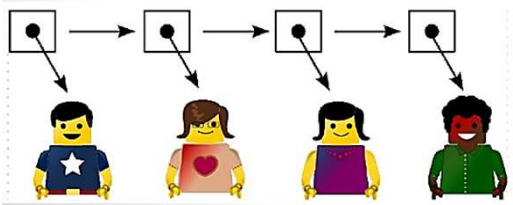
- **Esta aula foi baseada na Apostila de Estrutura de Dados – PUC-RIO**

  Profs. Waldemar Celes e José Lucas Rangel

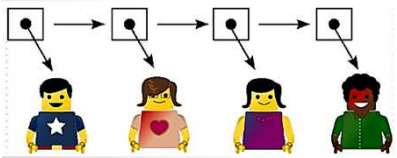
- **Vimos que Lista Encadeada**
  - É uma estrutura onde para cada novo elemento inserido, aloca-se um espaço de memória para armazená-lo.
  - O espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado.
  - **Não é possível garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo.**



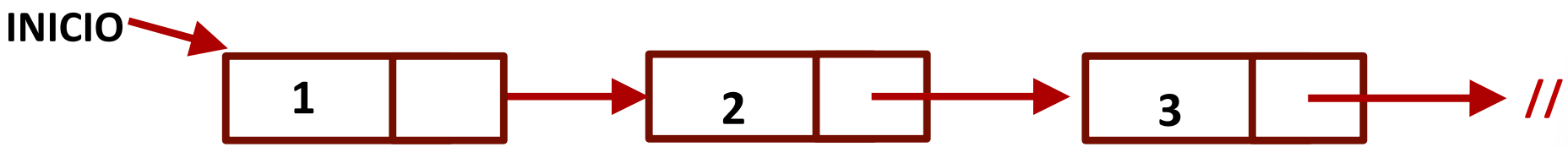
- **Vimos que Lista Encadeada**
  - **Não há acesso direto aos elementos da lista.**
  - Para que seja possível percorrer todos os elementos da lista, é necessário explicitamente guardar o encadeamento dos elementos,
  - O que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista

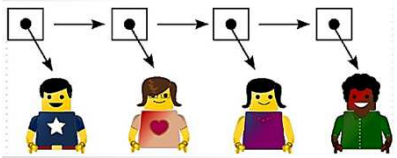


- **Vimos que Lista Encadeada**
  - A estrutura consiste numa sequência encadeada de elementos, em geral chamados de nós da lista.
  - A lista é representada por um ponteiro para o primeiro elemento (ou nó).
  - Do primeiro elemento, podemos alcançar o segundo seguindo o encadeamento, e assim por diante.
  - O último elemento da lista aponta para NULL, sinalizando que não existe um próximo elemento



- **Exemplo**

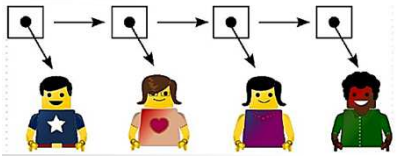




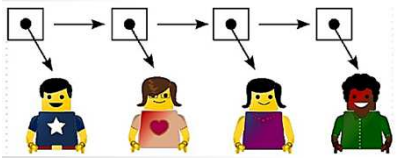
Listas Encadeadas

- **Estrutura**

```
struct lista
{
        int info;
        struct lista* prox;
};
typedef struct lista Lista;
```



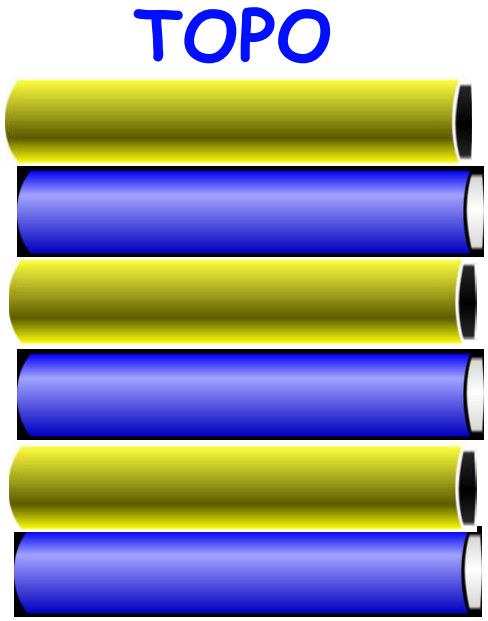
- **Listas Encadeadas**
  - Dentro deste conceito de lista temos 3 formas de manipulação:
    - Pilhas;
    - Filas;
    - E listas propriamente dita.



- **Pilhas**
  - Estrutura de dados **mais simples**
  - Possivelmente por essa razão, é a **estrutura de dados mais utilizada** em programação,
  - Sendo inclusive implementada diretamente pelo hardware da maioria das máquinas modernas

- **Pilha**
  - **Ideia Fundamental:**
    - 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

- **Pilha**
  - **Ideia Fundamental:**
    - **LIFO (“ *Last In First Out*”):**
      - O **Último** que **Entra** é o **Primeiro** que **Sai**

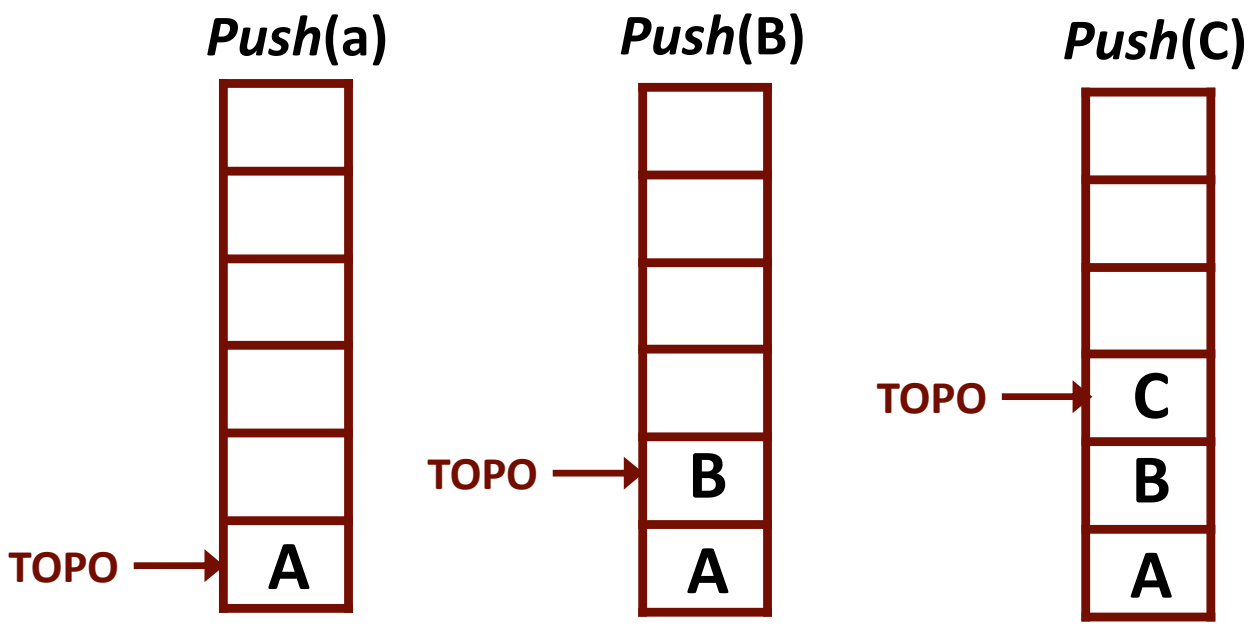


Analogia com uma pilha de livros:

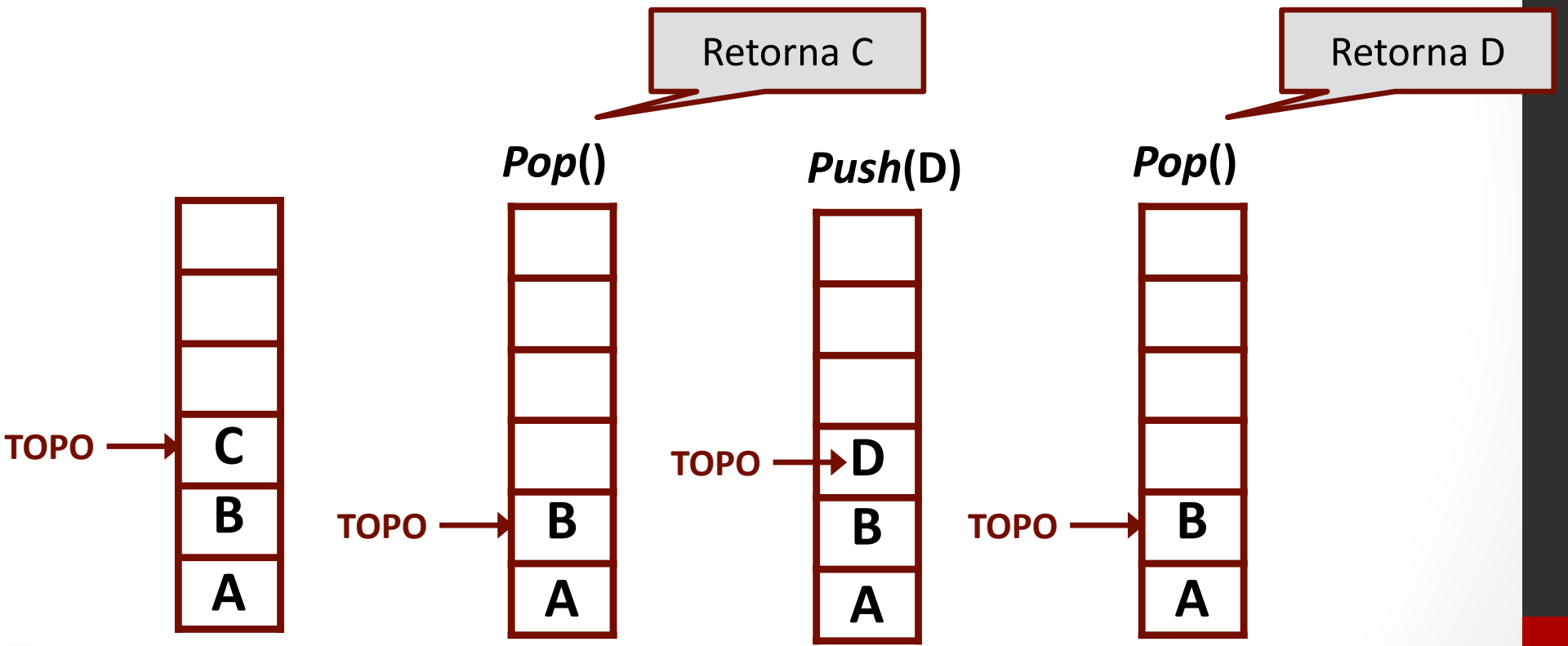
- Se quiser inserir um livro novo será no topo, em cima do amarelo
- Se quiser tirar o segundo livro, o azul, tenho que tirar o amarelo primeiro

- **Pilha**
  - Operações Básicas:
    - Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:
    - A operação para **Empilhar** (“***Push***”) um novo elemento, inserindo-o no topo, e
    - A operação para **Desempilhar** (“***Pop***”) um elemento, removendo-o do topo.

- **Pilha**
  - Funcionamento Conceitual de uma Pilha



- **Pilha**
  - Funcionamento Conceitual de uma Pilha



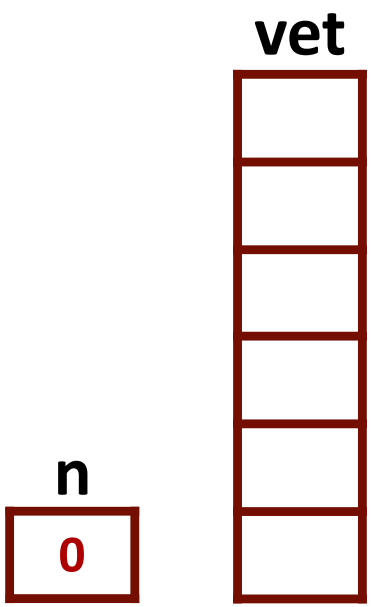
- **Pilha Implementação:**
  - Há duas implementações de pilha:
    - Usando vetor e
    - Usando Lista Encadeada

- **Pilha Implementação:**
  - **Independentemente do tipo da implementação, vamos considerar cinco operações:**
    - 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.

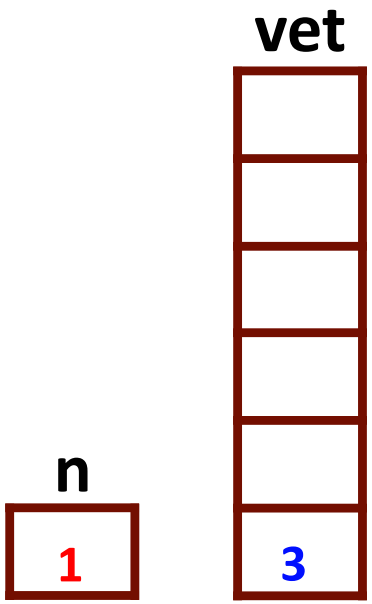
- **Implementação de Pilha Usando Vetor:**
  - A estrutura da pilha tem um **limite conhecido (tamanho do vetor)**
  - A implementação com vetor é bastante simples.
  - Há um vetor (vet) para armazenar os elementos da pilha.
  - Os elementos inseridos ocupam as primeiras posições do vetor.
  - Desta forma, há **n** elementos armazenados na pilha, o elemento vet[n-1] representa o elemento do topo.

- **Implementação de Pilha Usando Vetor:**
  - A estrutura que representa o tipo pilha deve, portanto, ser composta:
    - pelo vetor e
    - pelo número de elementos armazenados

```
#define MAX 6
struct pilha
 {
    int n;
    int vet[MAX];
};
```

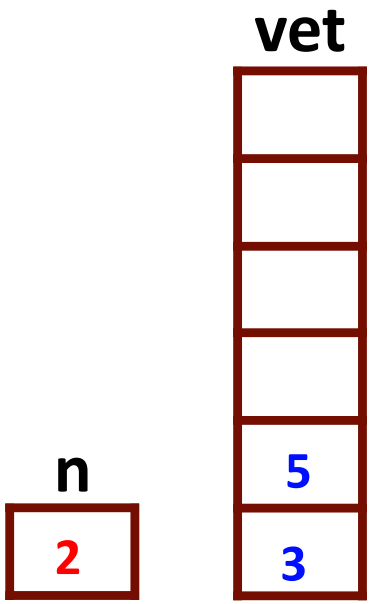


- **Implementação de Pilha Usando Vetor - Exe**



Inserir o Elemento 3

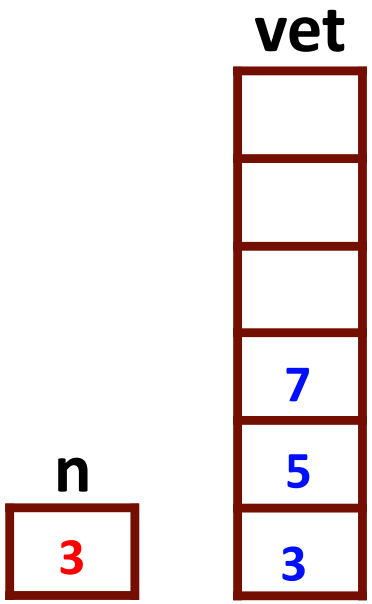
- **Implementação de Pilha Usando Vetor - Exe**



Inserir o Elemento 3

Inserir o Elemento 5

- **Implementação de Pilha Usando Vetor - Exe**

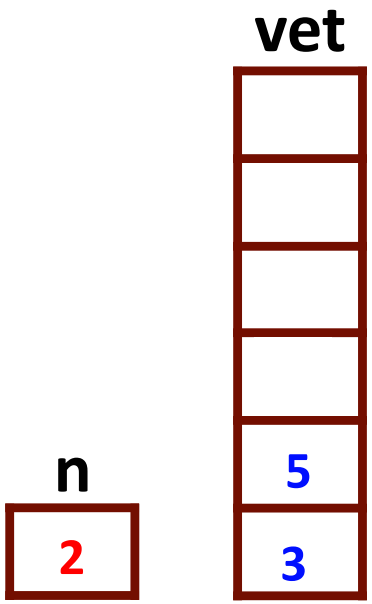


Inserir o Elemento 3

Inserir o Elemento 5

Inserir o Elemento 7

- **Implementação de Pilha Usando Vetor - Exe**



Inserir o Elemento 3

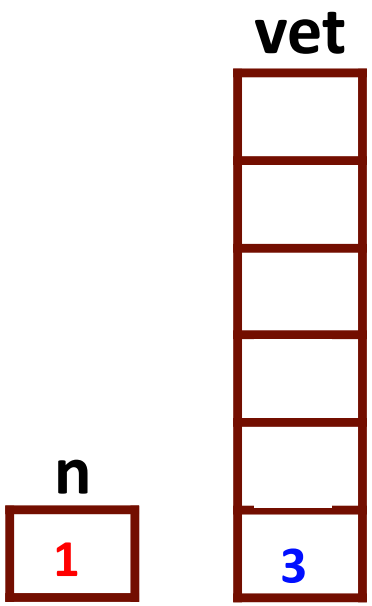
Inserir o Elemento 5

Inserir o Elemento 7

Remover um elemento

7 Removido

- **Implementação de Pilha Usando Vetor - Exe**



Inserir o Elemento 3

Inserir o Elemento 5

Inserir o Elemento 7

Remover um elemento

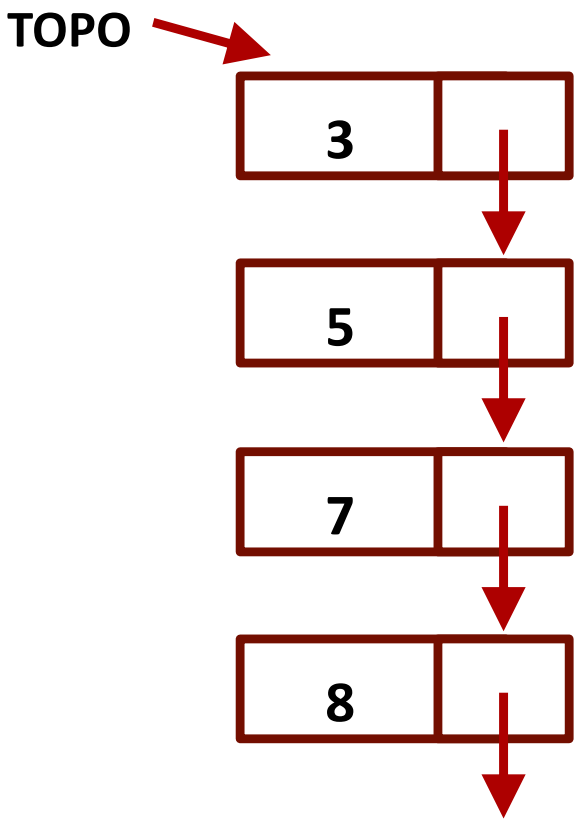
7 Removido

Remover um elemento

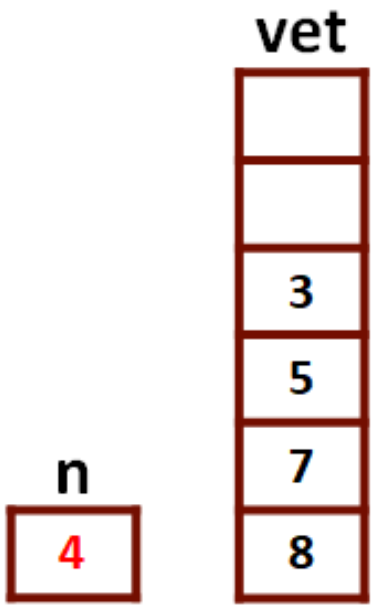
5 Removido

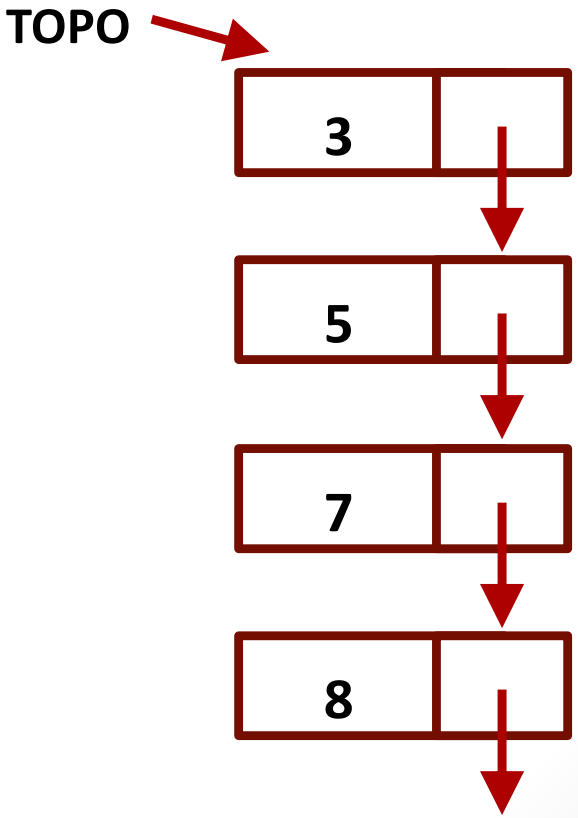
- **Implementação de Pilha em Lista Dinâmica**
  - **O número máximo de elementos** que serão armazenados na pilha **NÃO** é conhecido.
  - Vamos considerar das implementação de cinco mesmas operações:
    - Criar uma estrutura de pilha;
    - Inserir um elemento no topo (*Push*);
    - Remover o elemento do topo (*Pop*);
    - Verificar se a pilha está vazia;
    - Liberar a estrutura de pilha.

- **Implementação de Pilha em Lista Dinâmica**
  - **Para facilitar a compreensão vamos desenhar a pilha dinâmica como se fosse uma pilha propriamente dita**



- **Observe que**
  - O usuário do programa não consegue perceber que tipo de TAD programa está usando, aliás que o programa usa TAD's





- **Implementação de Pilha em Lista Dinâmica**
  - O nó da lista para armazenar valores inteiros pode ser dado por:

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

- **Implementação de Pilha em Lista Dinâmica**
  - A estrutura da pilha é então simplesmente:

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

Estrutura que armazena o endereço inicial da Pilha, ou seja, o topo

- **Implementação de Pilha em Lista Dinâmica**
  - **O primeiro elemento da lista representa o topo da pilha**.
  - Cada novo elemento **é inserido no início da lista** e, consequentemente,
  - Sempre que solicitado, **retira-se o elemento também do início da lista**

- **A função Cria**
  - É a função que inicializa o estrutura da Pilha para que o programa possa trabalhar corretamente com a mesma

```
Pilha* CRIA ()
{
    Pilha* p;
    p=(Pilha*)malloc(sizeof(Pilha));
    p->Topo = NULL;
  return p;
}
void main()
{
    Pilha* P;
    P = CRIA();
}
```

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

- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

```
Pilha* CRIA ()
{
    Pilha* p;
    p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```

MEMÓRIA

- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

```
Pilha* CRIA ()
{
    Pilha* p ;
    p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```

MEMÓRIA

PILHA1

LIXO

E20

- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

```
Pilha* CRIA ()
{
    Pilha* p ;
    p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```

MEMÓRIA

PILHA1

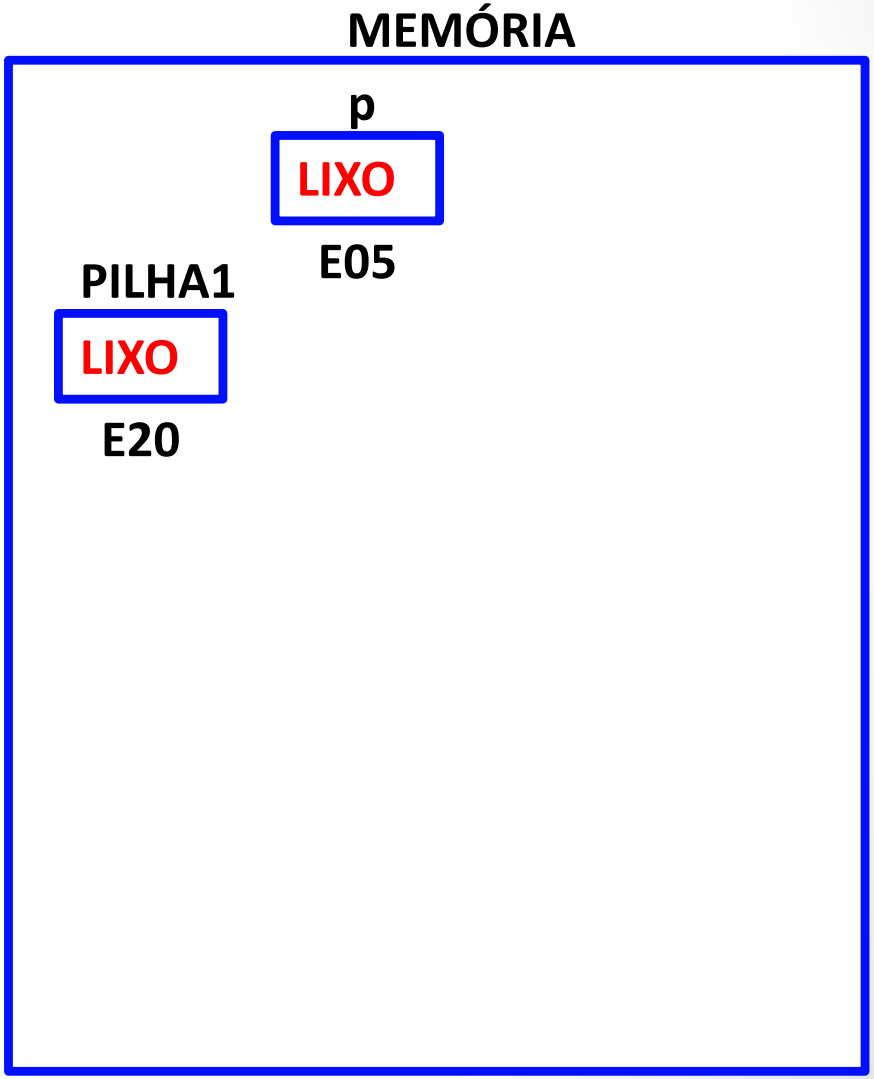
LIXO

E20

- **Implementação de Pilha em Lista**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

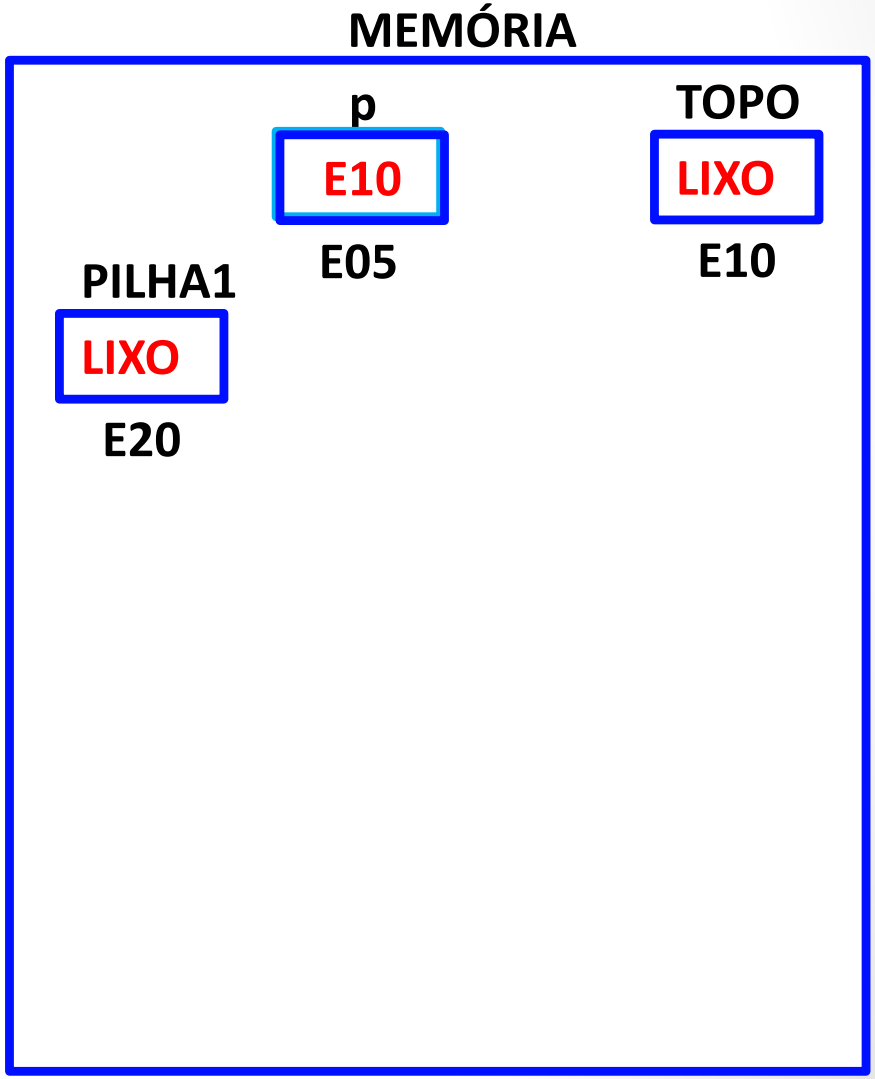
```
Pilha* CRIA ()
{
    Pilha* p ;
     p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```



- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

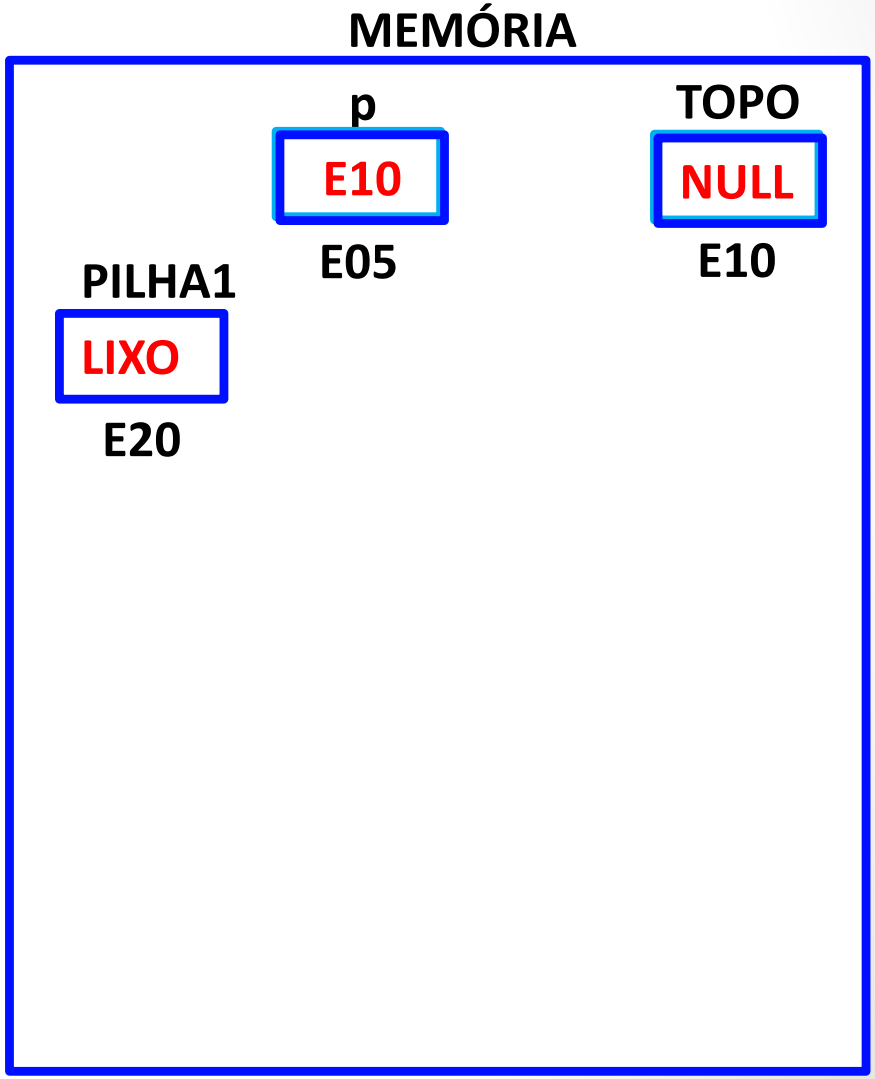
```
Pilha* CRIA ()
{
    Pilha* p ;
     p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```



- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

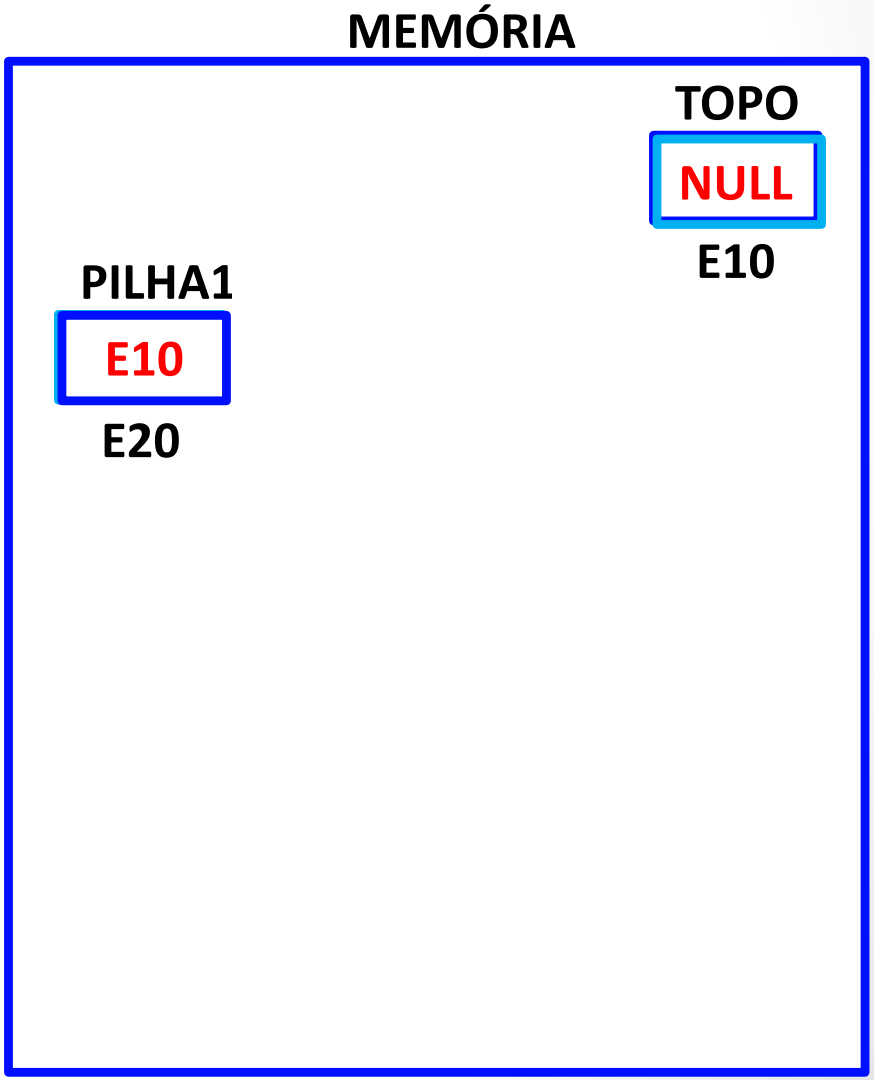
```
Pilha* CRIA ()
{
    Pilha* p ;
     p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
   return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```



- **Implementação de Pilha em Lista Dinâmica**

  - A função ***CRIA*** aloca a estrutura da pilha e inicializa a pilha como sendo vazia.

```
Pilha* CRIA ()
{
    Pilha* p ;
     p=(Pilha*)malloc(sizeof(Pilha));
    p->TOPO = NULL;
  return p;
}
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
}
```



Pilhas

- **A função push()**
  - É a função de inserção de um elemento na pilha
  - Esse elemento é inserido **no topo (início)** da pilha e o topo passa a apontar para ele

- **A função pop()**
  - É a função de remove um elemento da pilha
  - A função retorna o elemento removido
  - Esse elemento é removido do topo (início) da pilha e o topo passa a apontar para o próximo

Pilhas

- **Exercícios**

1. .

Considerando as funções/procedimentos abaixo

```
Pilha* CRIA (void);
void push (Pilha* p, int v)
int pop (Pilha* p)
```

Faça o teste de mesa para o programa abaixo

```
int main()
{
    Pilha *um, *dois; int a;
    um=CRIA();  dois=CRIA();
    push(um,1)
    push(dois,3)
    push(um,4)
    push(dois,5)
     push(um,6)
    push(dois,7)
    a=pop(dois)
    push(dois,pop(um))
    a=pop(dois)
    push(dois,pop(dois))
    push(dois,pop(um))
    push(dois,pop(dois))
    push(um,pop(um))
    push(um,pop(dois))
}
```

- **Exercícios**

2. Observe os quadros I e II, relacionados à estrutura de dados pilha

QUADRO I – Operações permitidas pela estrutura de dados pilha

Operação	Descrição
Push(PILHA,w)	• *Insere um elemento w na pilha*
Pop(PILHA)	• *Remove o elemento de topo da pilha*
Top(PILHA)	• *Acessa, sem remover, o elemento de topo da pilha*

QUADRO II – operações realizadas

a) *Push(STE, SANTO_ANTÔNIO)*
b) *Push(STE, SANTA_FILOMENA)*
c) *Pop(STE)*
d) *Push(STE, SANTO_AGOSTINHO)*
e) *Top(STE)*
f) *Push(STE, SANTA_CATARINA)*
g) *Top(STE)*
h) *Push(STE, SANTO_EXPEDITO)*
i) *Pop(STE)*
j) *Push(STE, Top(STE))*
k) *Push(STE, Pop(STE))*
l) *Push(STE, SANTA_GENOVEVA)*
m) *Pop(STE)*
n) *Push(STE, Top(STE))*

Determine o elemento de topo da pilha, após a execução de todas as operações indicadas no **quadro II**

- **A função push()**
  - É a função de inserção de um elemento na pilha
  - Esse elemento é inserido **no topo (início)** da pilha e o topo passa a apontar para ele
  - Sintaxe da função:

```
void push (Pilha* p, int v)
```

Função que não retorna nada

Passa uma estrutura do tipo pilha para ser atualizada

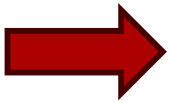
Valor a ser inserido na pilha

- **A função push()**

Dado as Estruturas:

```
struct pilha
{
        No* TOPO;
};

typedef struct pilha Pilha;
```



```
struct no
{
  float info;
  struct no* prox;
};

typedef struct no No;
```

```
/* função inserir um elemento na pilha*/

    void push (Pilha* p, float v)
    {
            p->TOPO = ins_ini(p->TOPO,v);
    }
```

- **A função push()**

```
/* função inserir um elemento na pilha*/

    void push (Pilha* p, float v)
    {
            p->TOPO = ins_ini(p->TOPO,v);
    }
```

```
/* função auxiliar: insere no início */
        No* ins_ini (No* t, float a)
        {
          No* p = (No*) malloc(sizeof(No));
          p->info = a;
          p->prox = t;
          return p;
        }
```

Pilhas

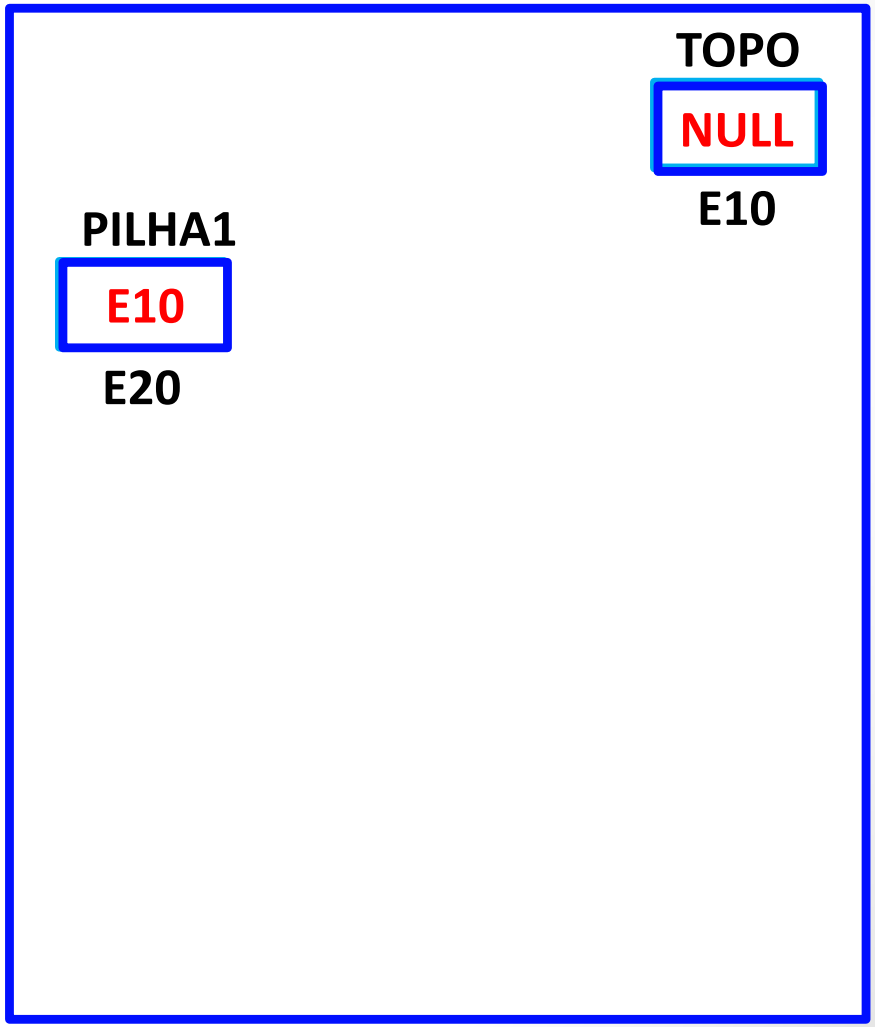
- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

¦
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```

MEMÓRIA



Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

¦
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```

MEMÓRIA

TOPO

NULL

E10

PILHA1

E10

E20

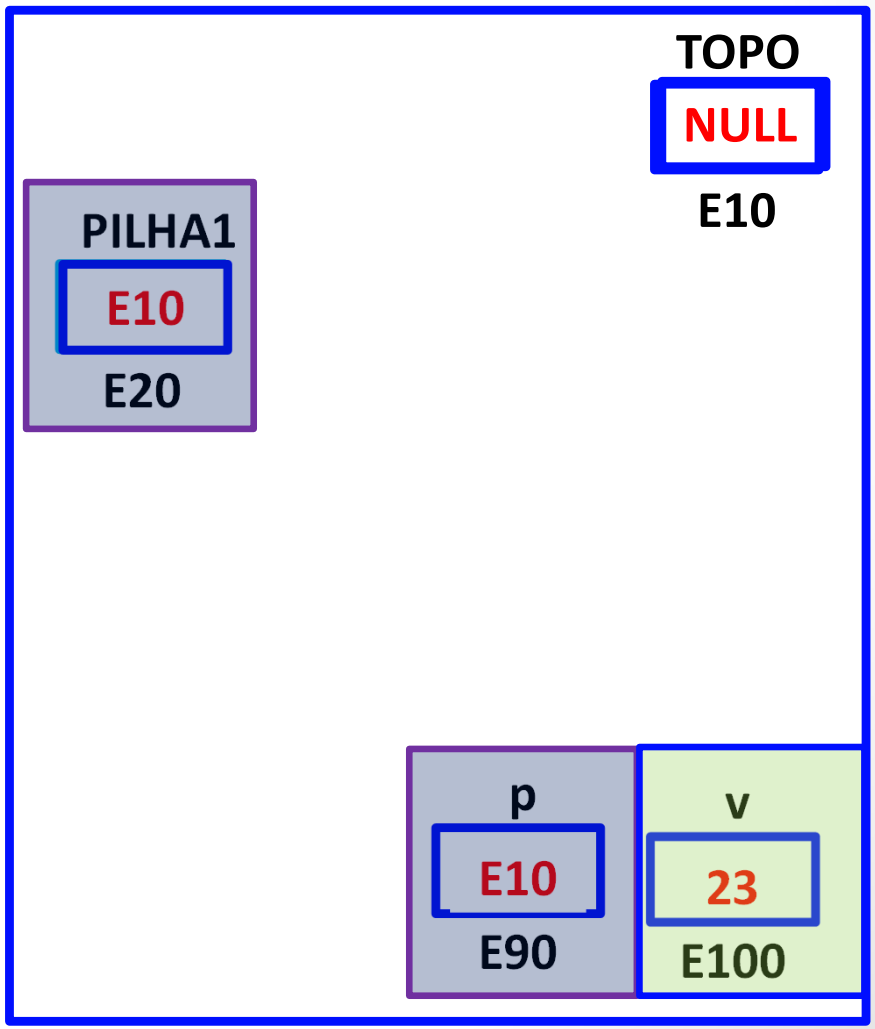
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

void main()
{
    Pilha *PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```

MEMÓRIA

TOPO
NULL
E10

PILHA1
E10
E20

p
E10
E90

v
23
E100

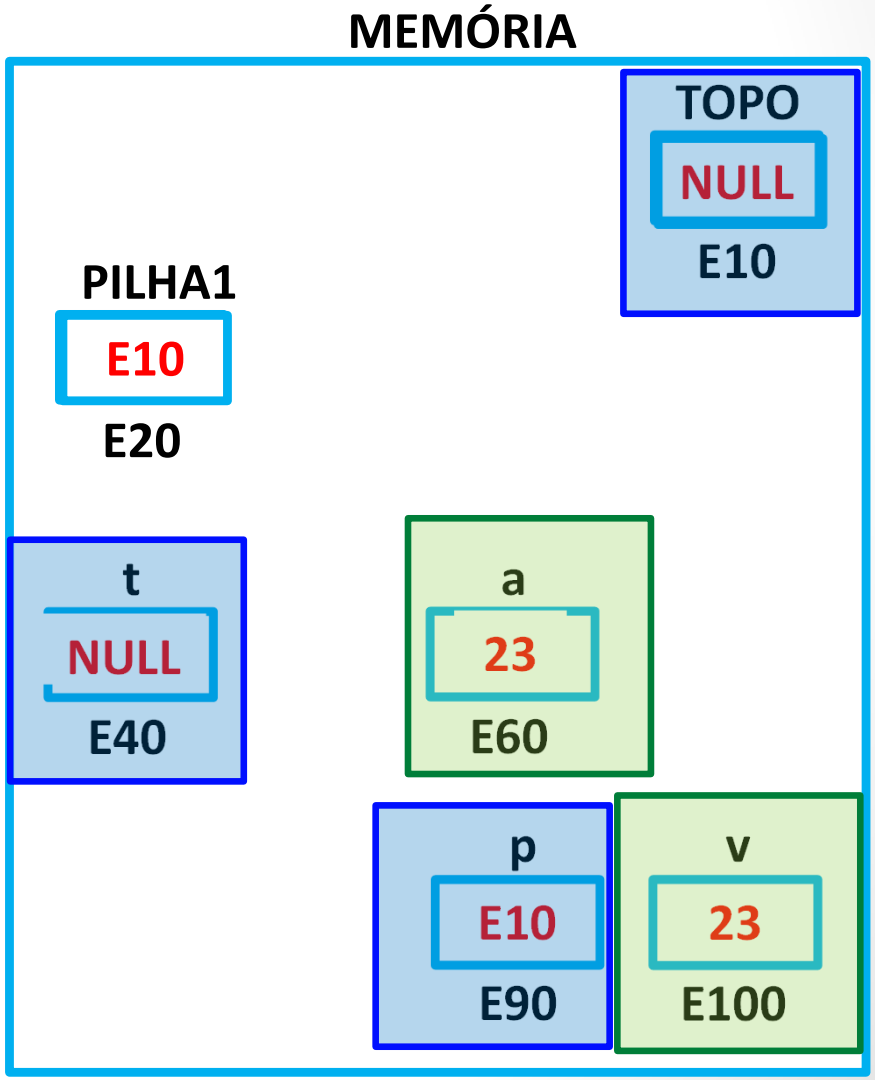
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO, v);
}

void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



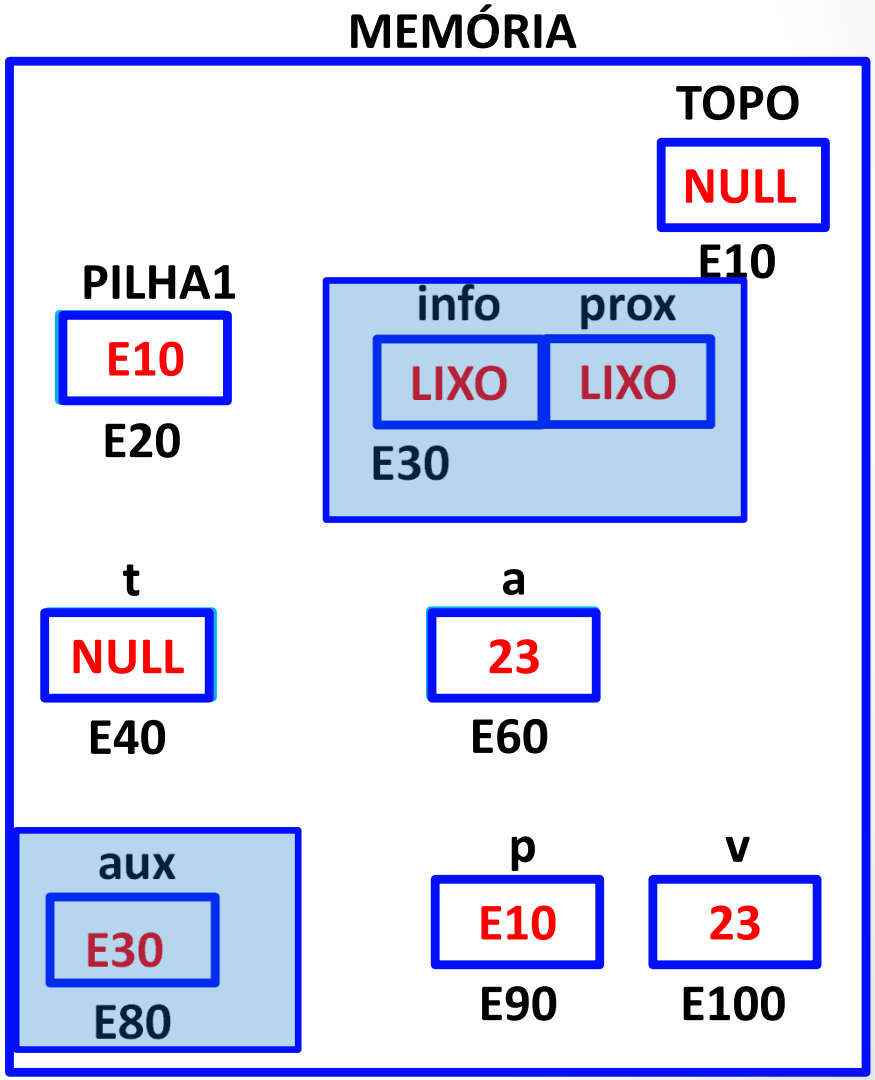
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

¦
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



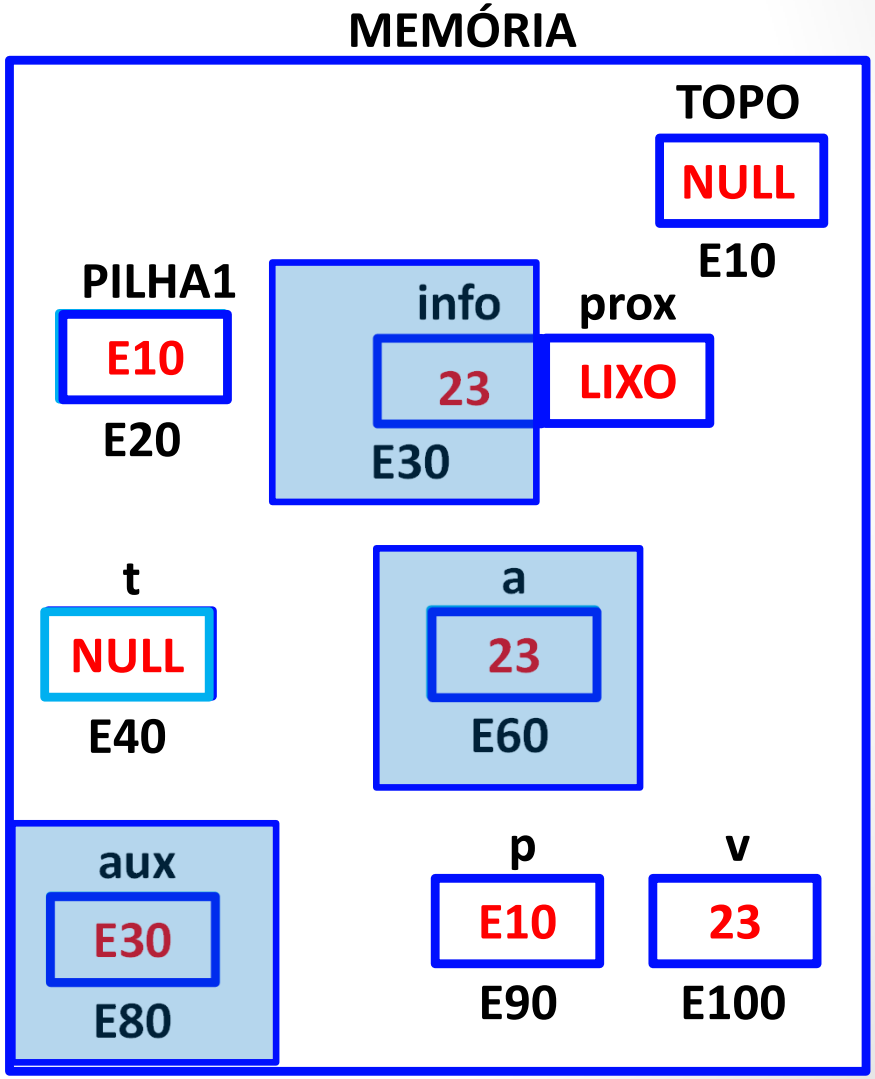
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

¦
void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



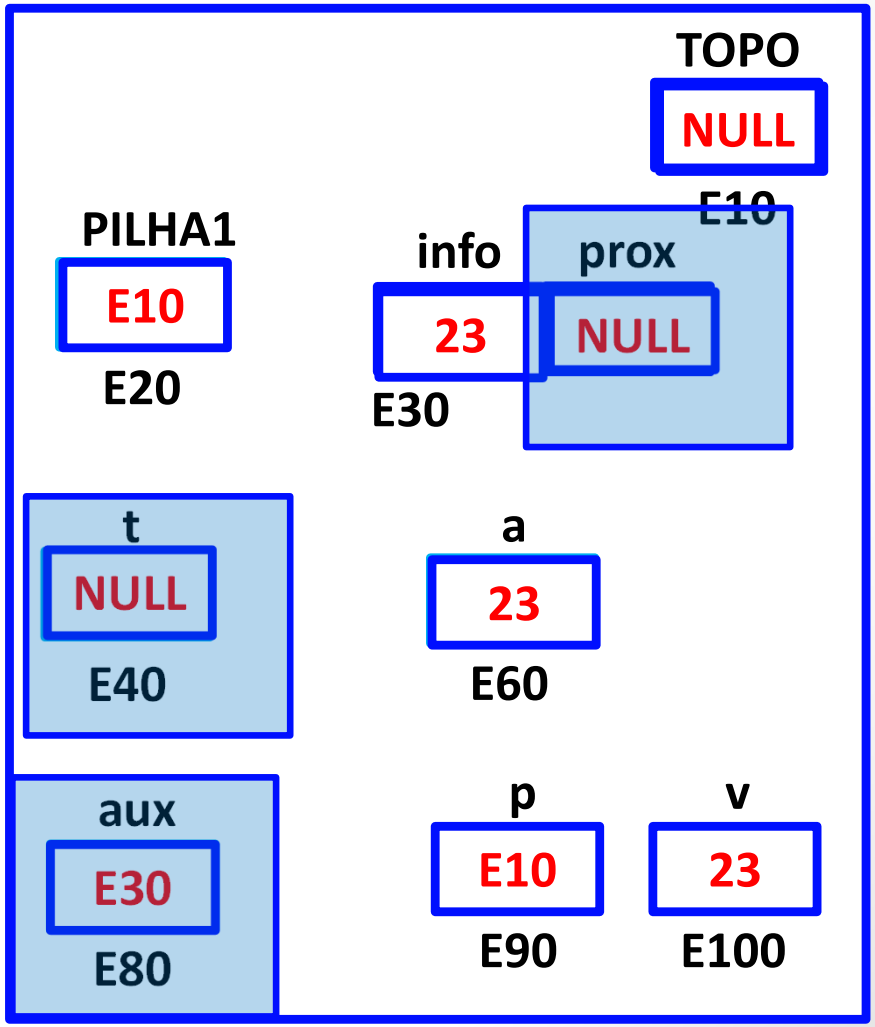
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



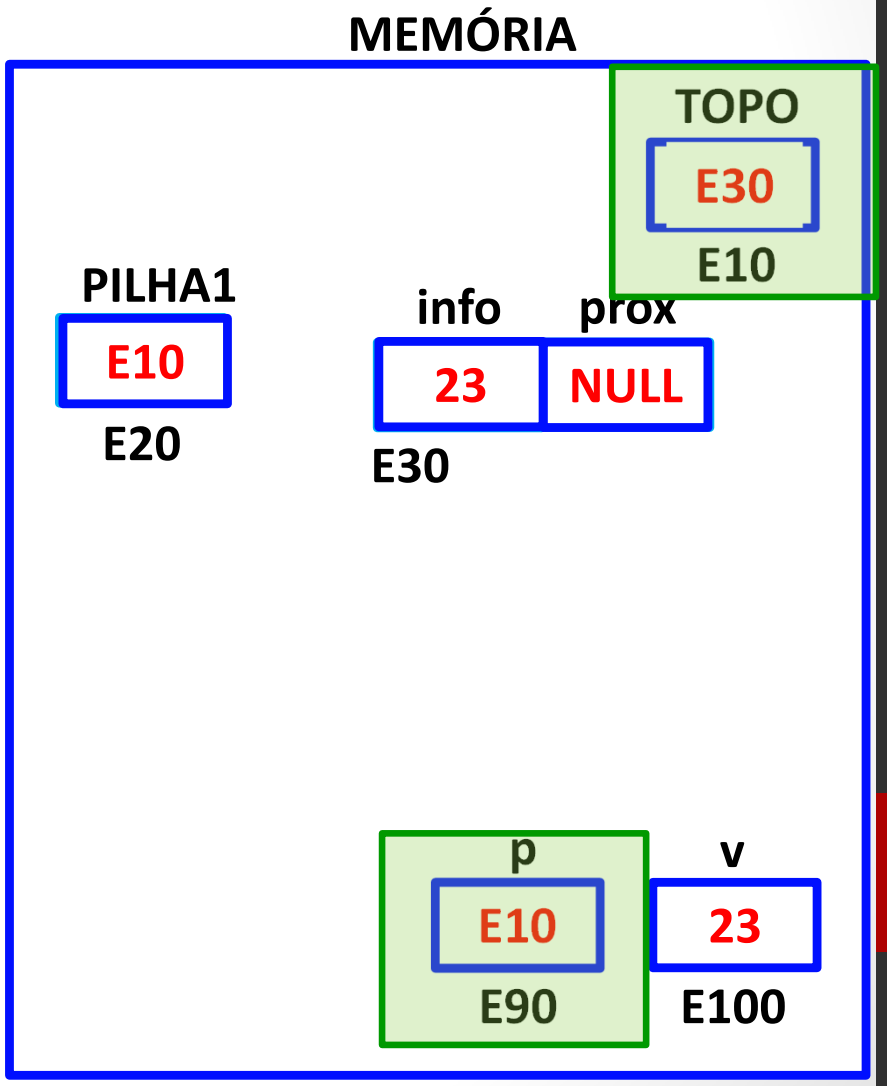
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



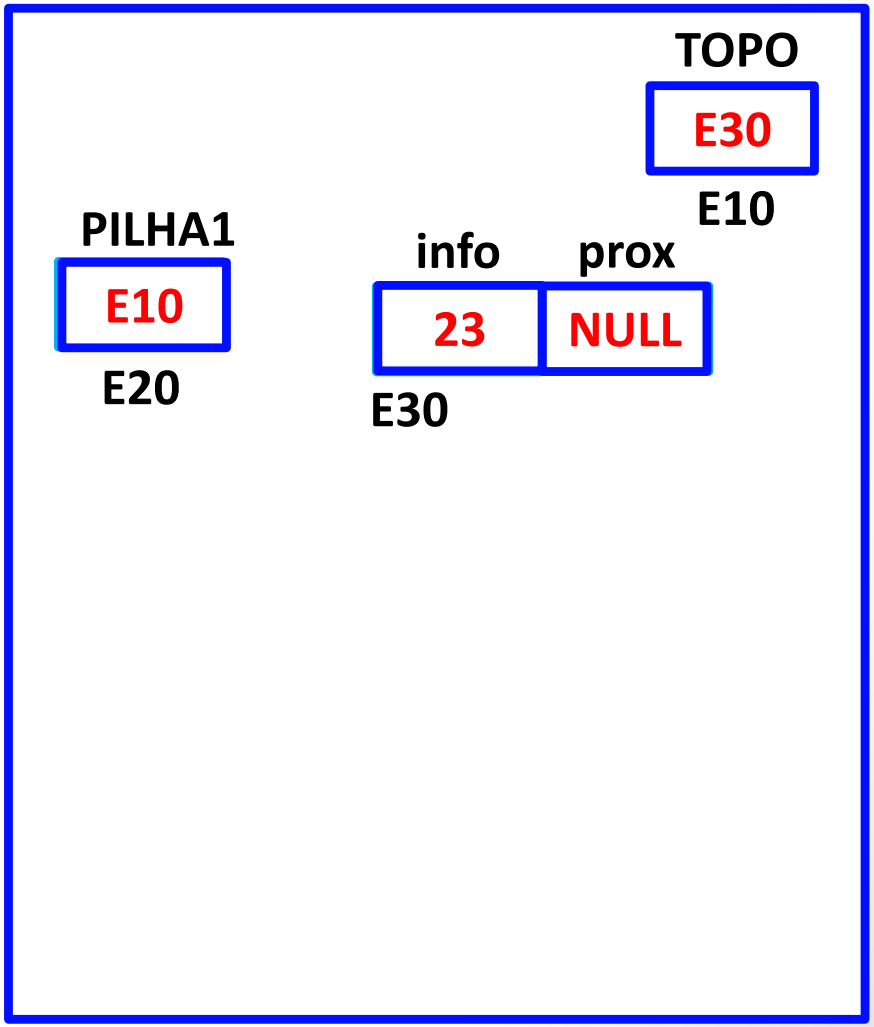
Pilhas

- **A função push()**

```
No* ins_ini (No* t, float a)
{
   No* aux = (No*) malloc(sizeof(No));
   aux->info = a;
   aux->prox = t;
   return aux;
}

void push (Pilha* p, float v)
{
   p->TOPO = ins_ini(p->TOPO,v);
}

void main()
{
    Pilha* PILHA1;
    PILHA1 = CRIA();
    push(PILHA1,23);
}
```



- **A função pop()**
  - É a função de remove um elemento da pilha
  - A função retorna o elemento removido
  - Esse elemento é removido do topo (início) da pilha e o topo passa a apontar para o próximo
  - Sintaxe da função:

```
int pop (Pilha* p)
```

Função que retorna o valor removido da pilha

Passa uma estrutura do tipo pilha para ser atualizada

- **A função pop()**

Dado as Estruturas:

```
struct pilha
{
        No* TOPO;
};

typedef struct pilha Pilha;
```

```
struct no
{
   float info;
   struct no* prox;
};

typedef struct no No;
```

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

Pilhas

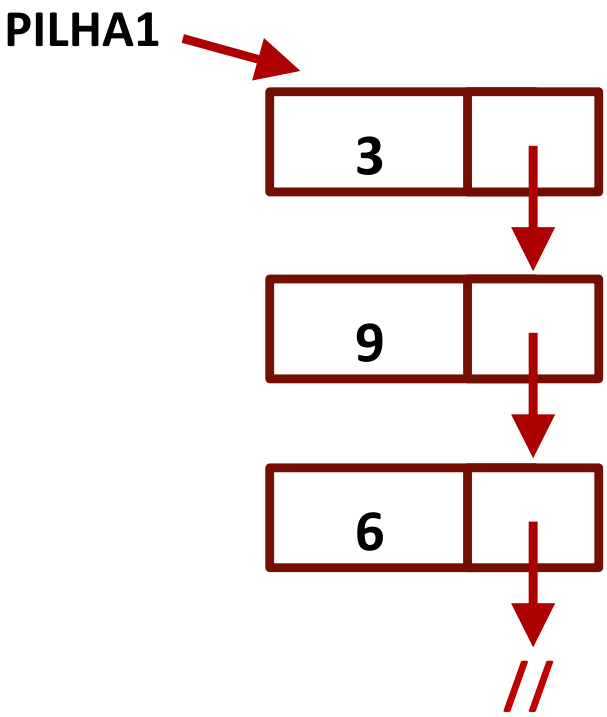
- **A função push()**

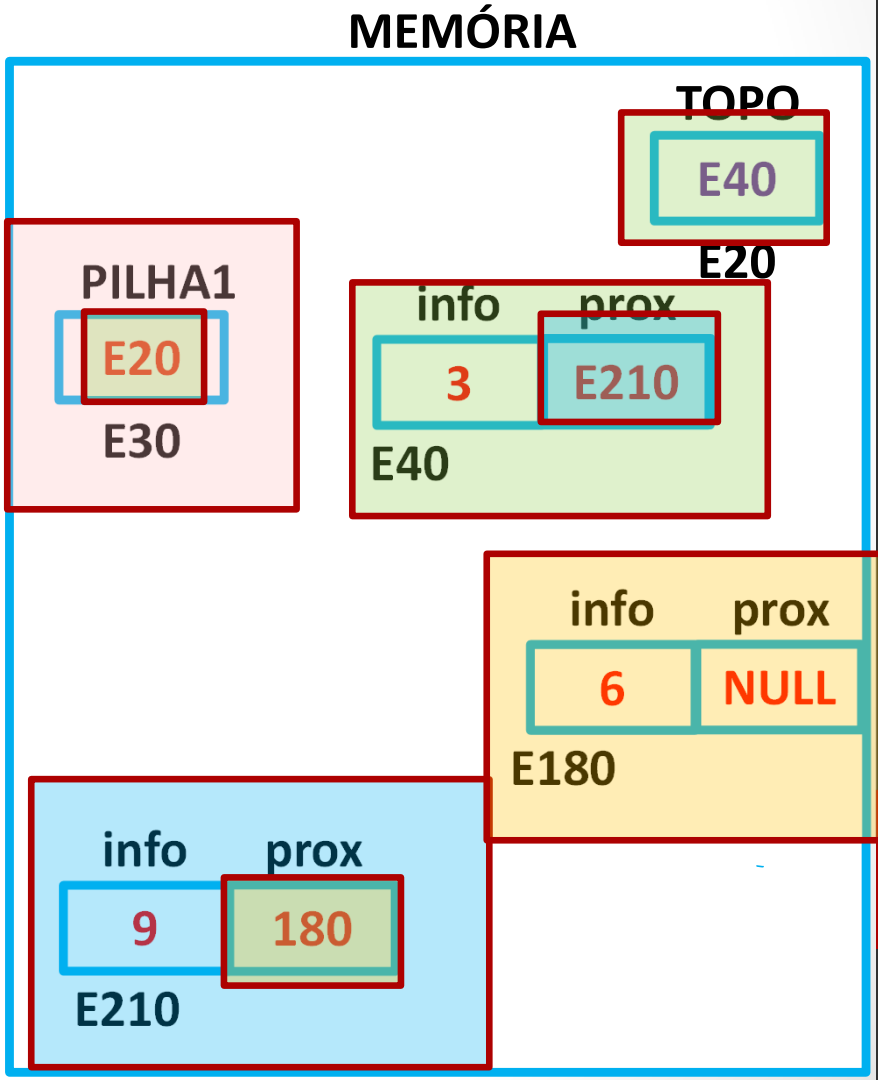
```
float pop (Pilha* p)
{
    float v;
    if (vazia(p))
    {
        printf("Pilha vazia.\n");
        exit(1); /* aborta programa
        */
    }
    v = p->TOPO->info;
    p->TOPO = ret_ini(p->TOPO);
    return v;
}
```

```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

Pilhas

- **A função pop()**



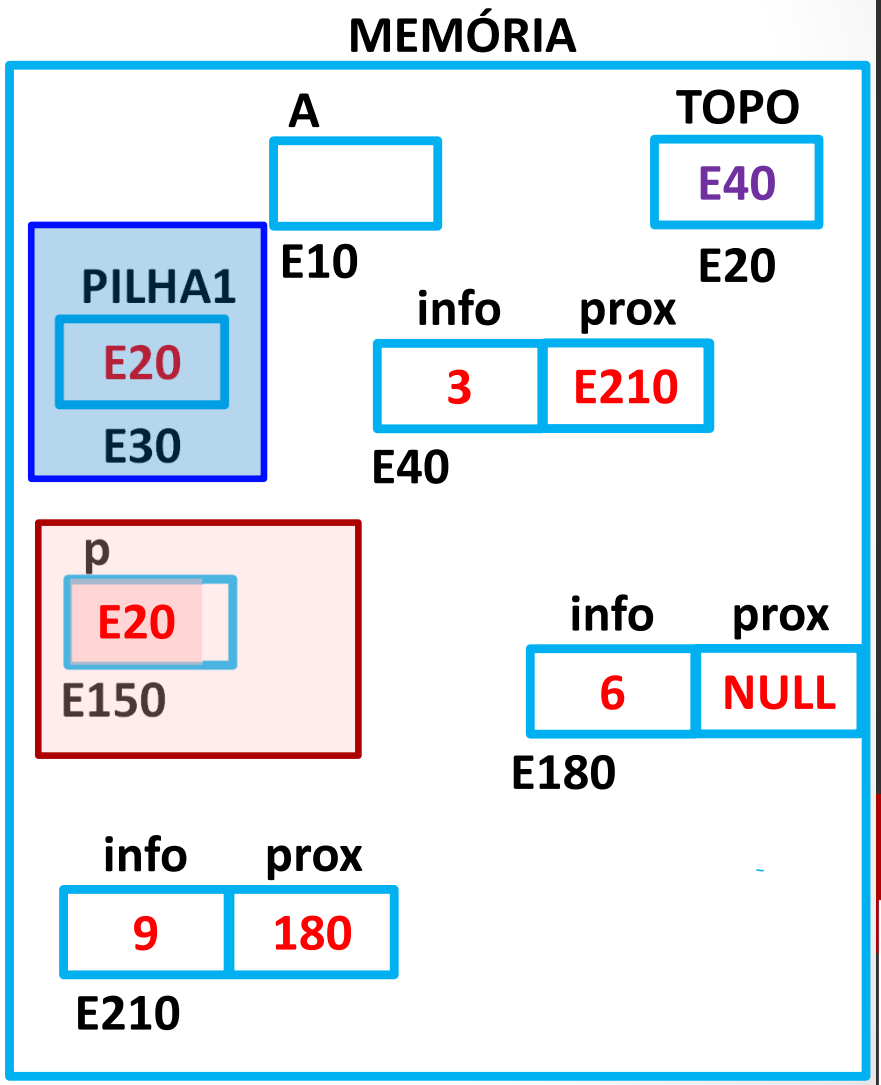


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(PILHA1);

- **A função pop()**

```
float pop (Pilha* p)
{
    float v;
    if (vazia(p))
    {
        printf("Pilha vazia.\n");
        exit(1); /* aborta programa
        */
    }
    v = p->TOPO->info;
    p->TOPO = ret_ini(p->TOPO);
    return v;
}
```

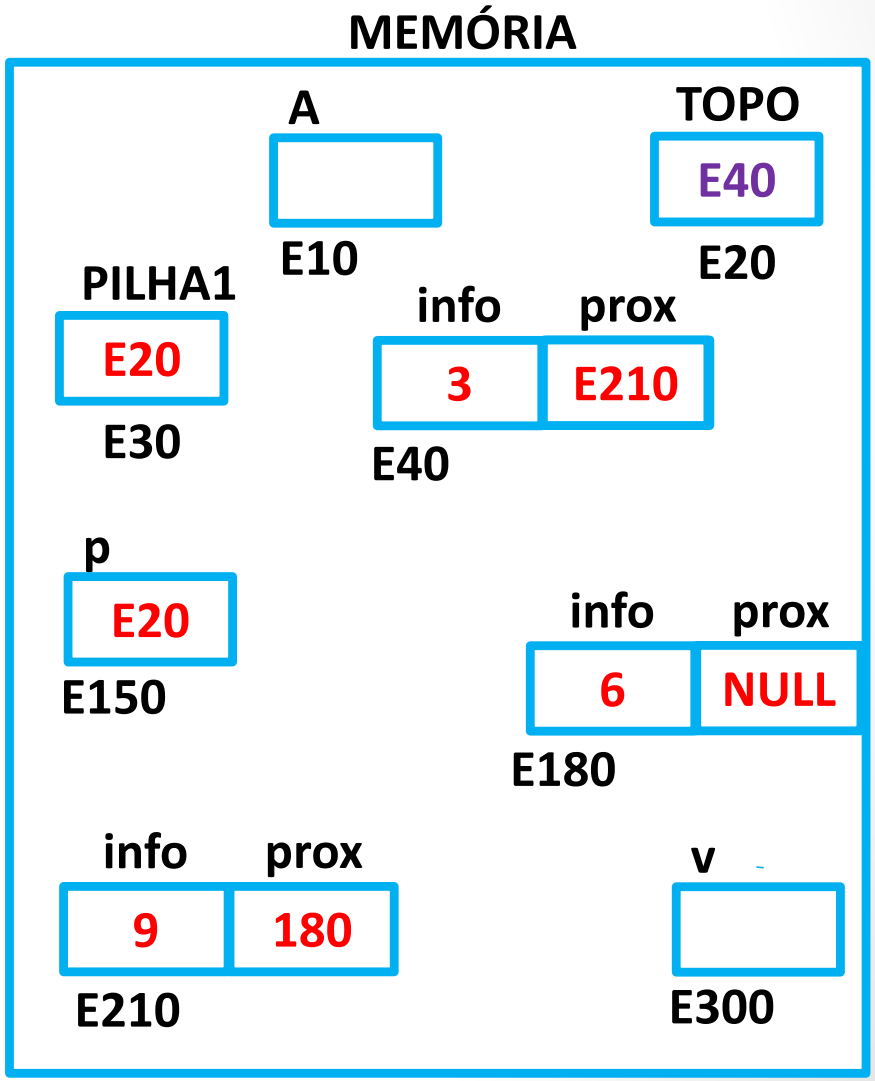


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(PILHA1);

- **A função pop()**

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

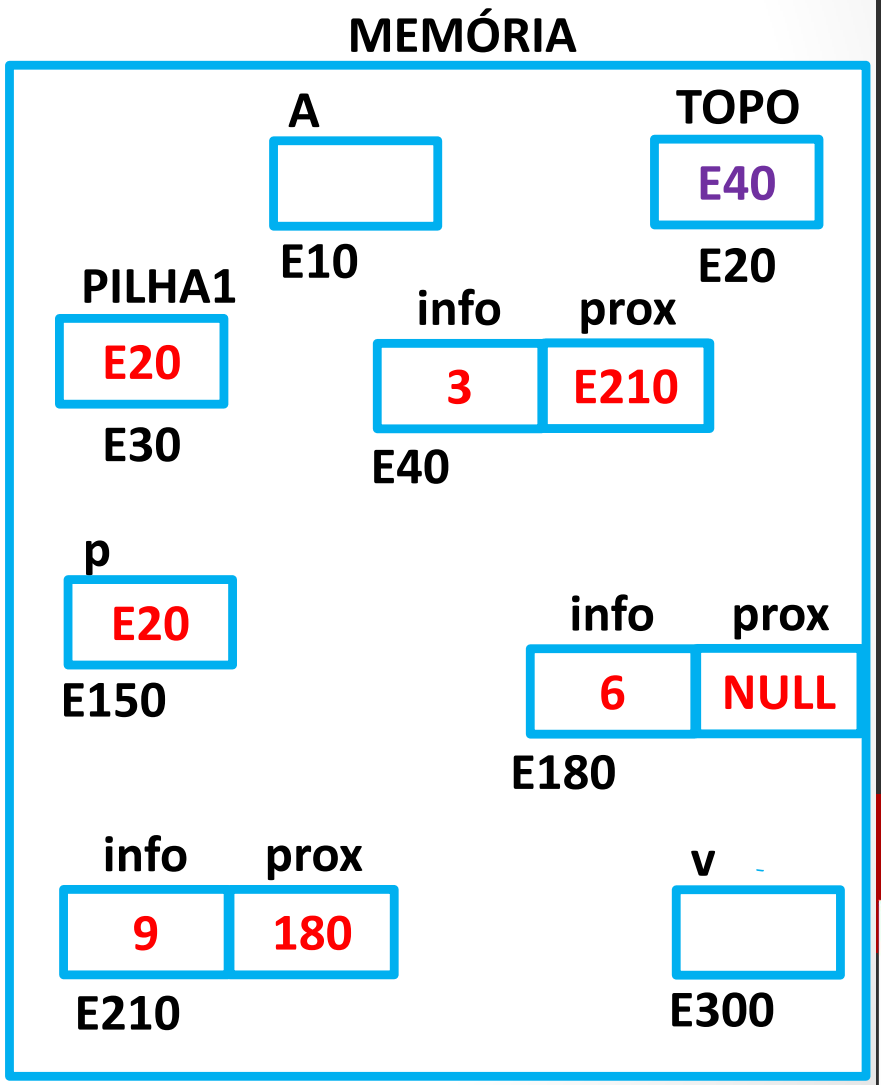


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(pilha1);

- **A função pop()**

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

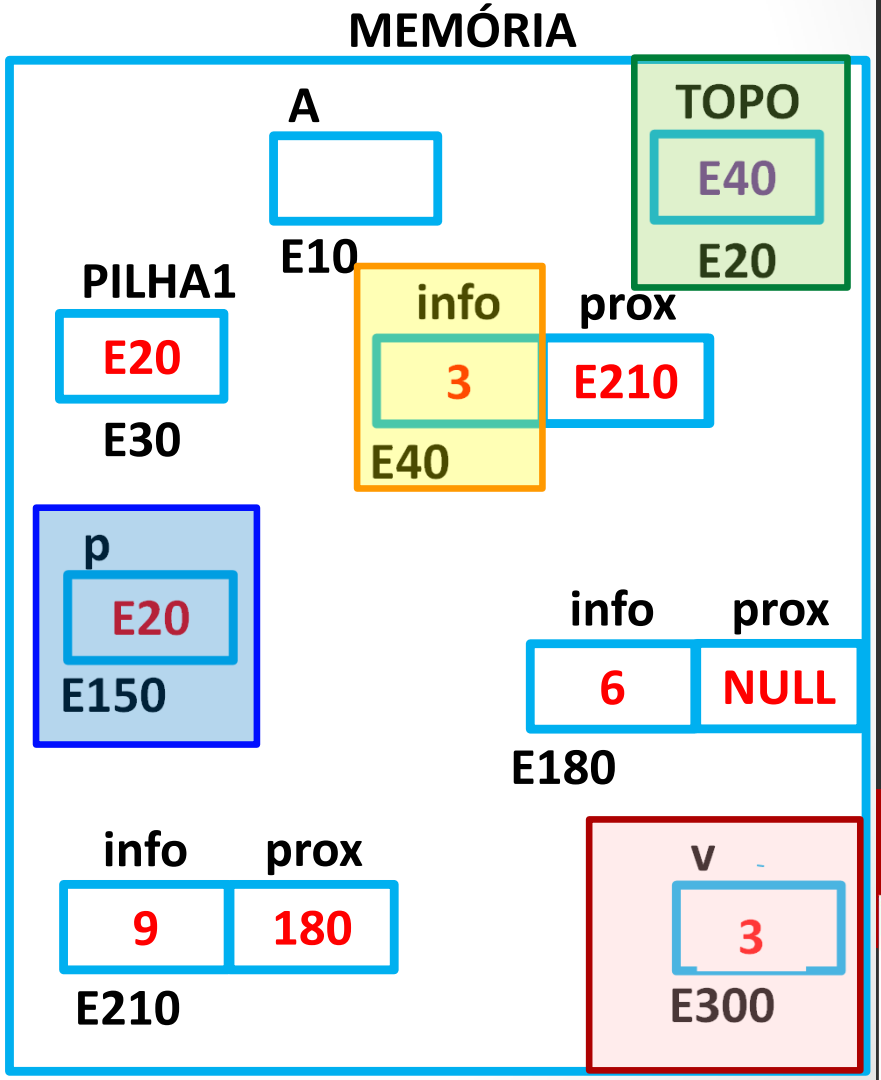


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(pilha1);

- A função pop()

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

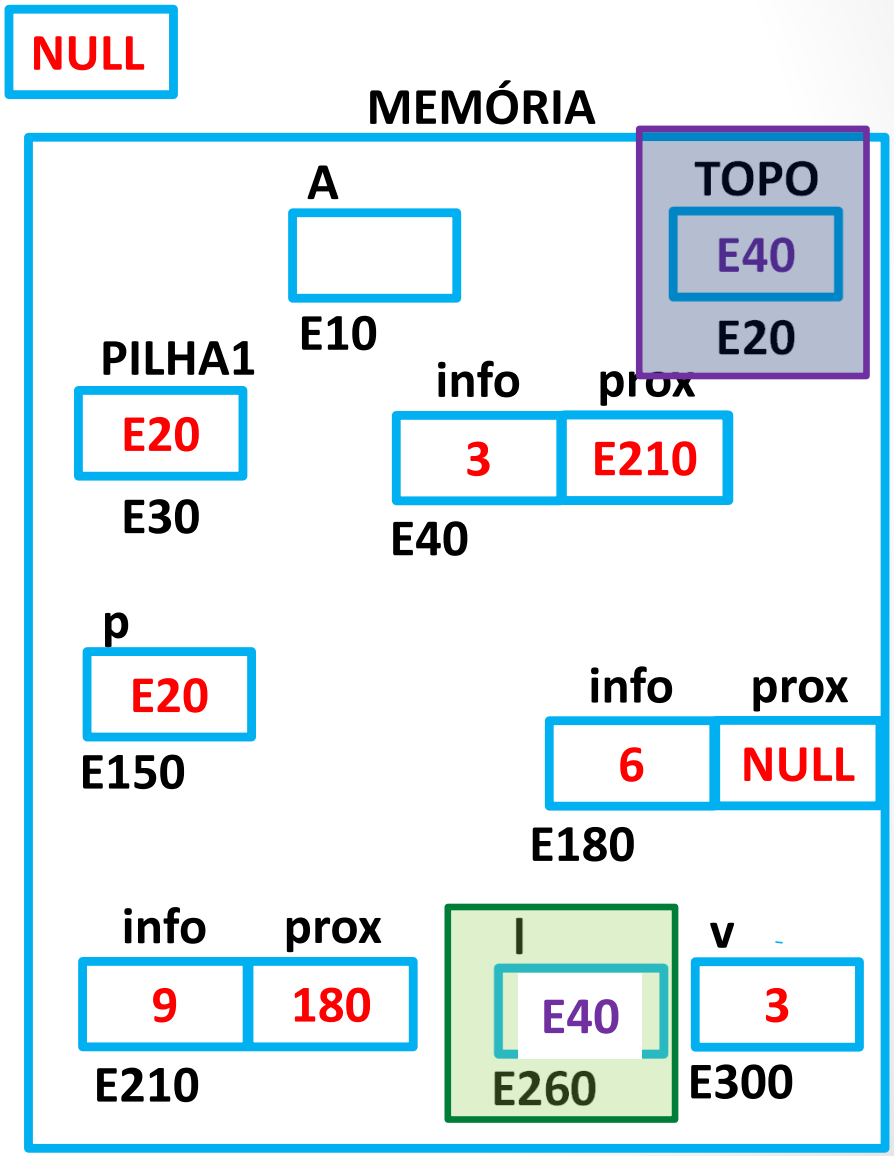


```
/* função auxiliar: retira no início */
No* ret_ini (No* l)
{
  No* p = l->prox;
  free(l);
  return p;
}
```

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

float A = pop(pilha1);

- A função pop()

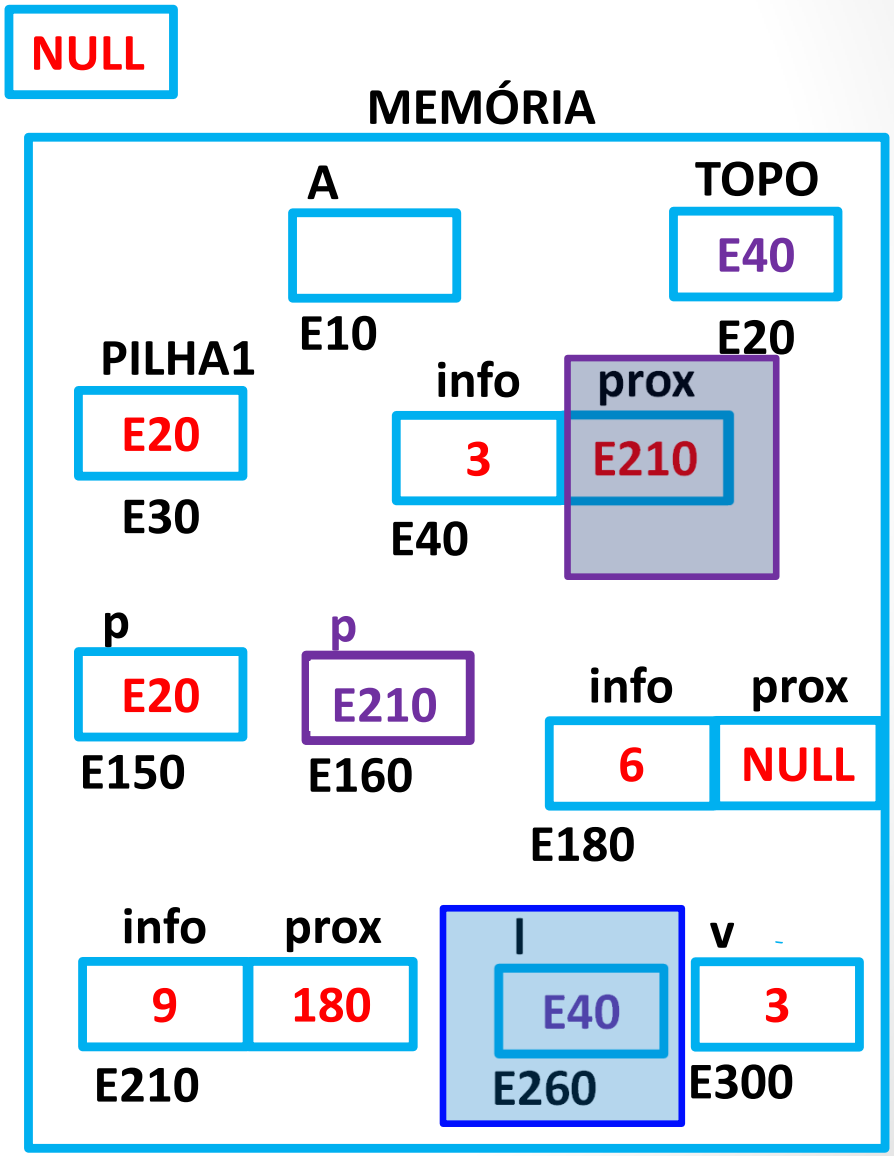


```
/* função auxiliar: retira no início */
No* ret_ini (No* l)
{
  No* p = l->prox;
  free(l);
  return p;
}
```

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

float A = pop(pilha1);

- **A função pop()**

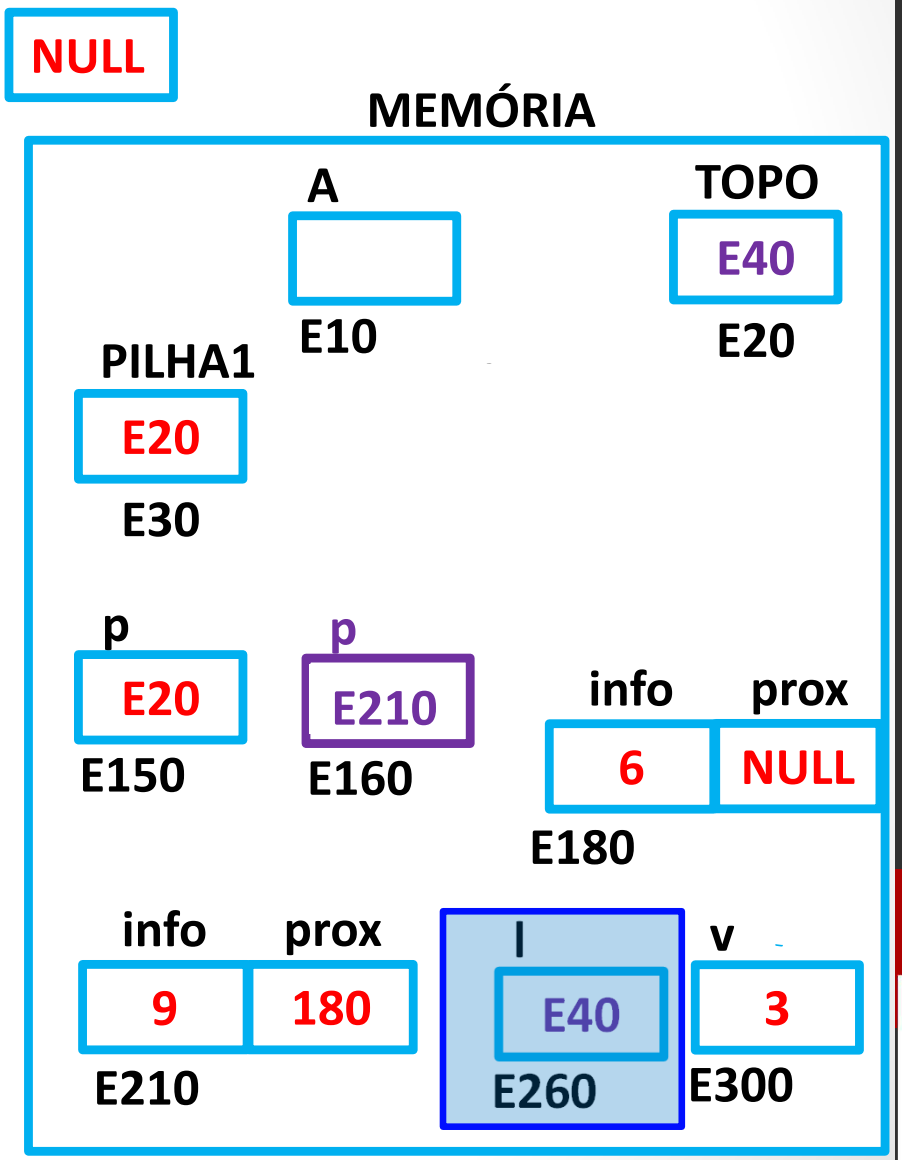


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```

float A = pop(pilha1);

- A função pop()

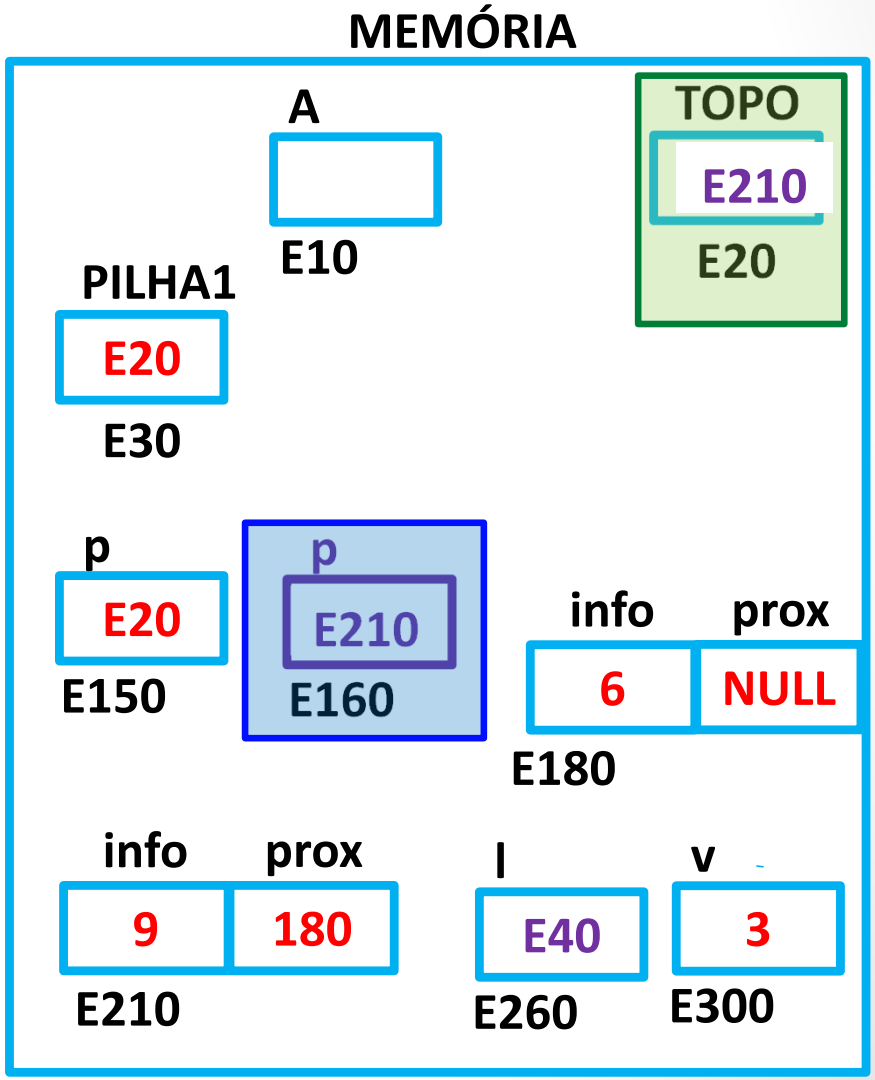


```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(pilha1);

- A função pop()

```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```



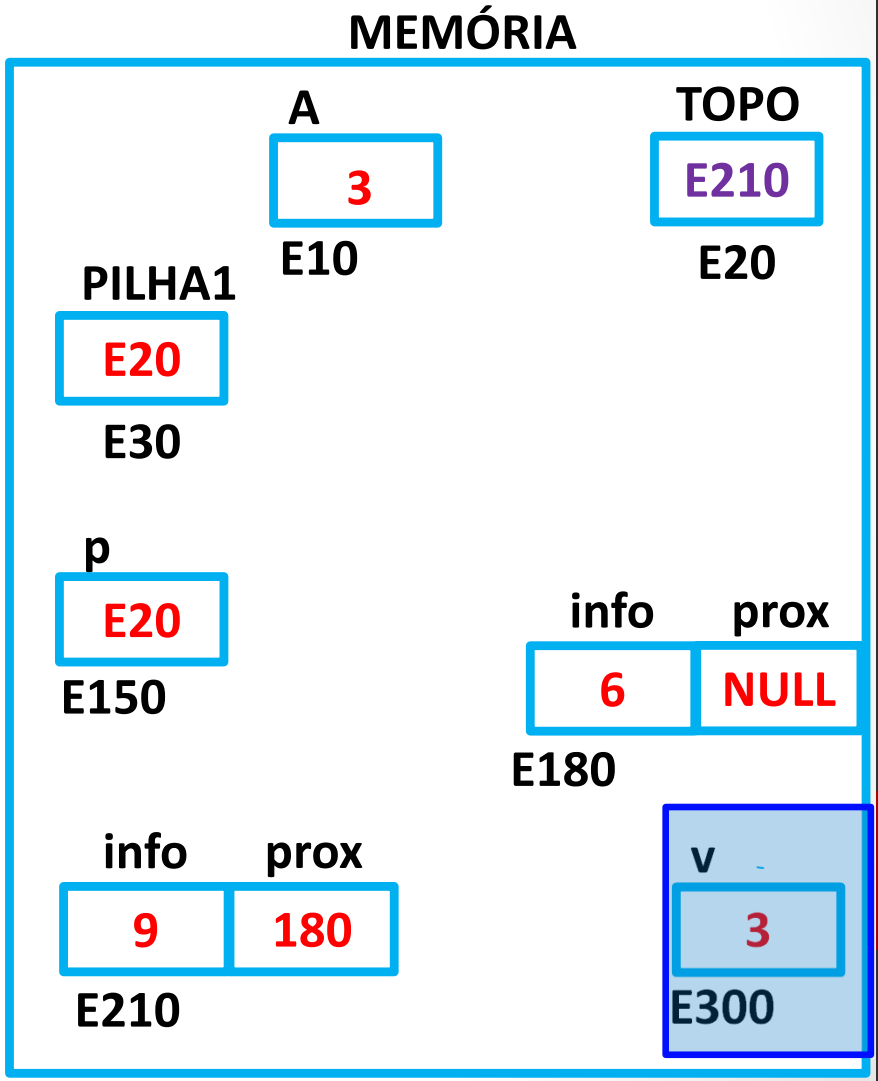
Pilhas

```
/* função auxiliar: retira no início */
        No* ret_ini (No* l)
        {
          No* p = l->prox;
          free(l);
          return p;
        }
```

float A = pop(pilha1);

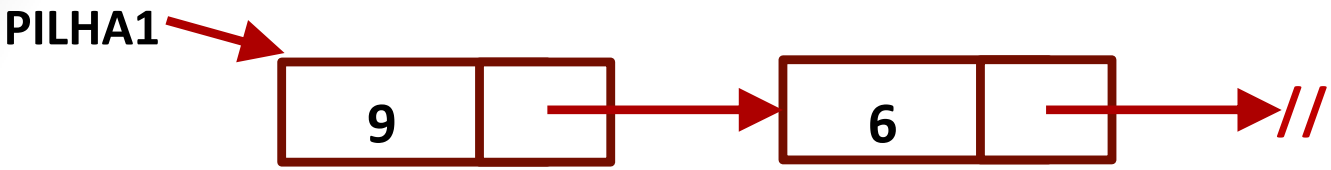
- A função pop()

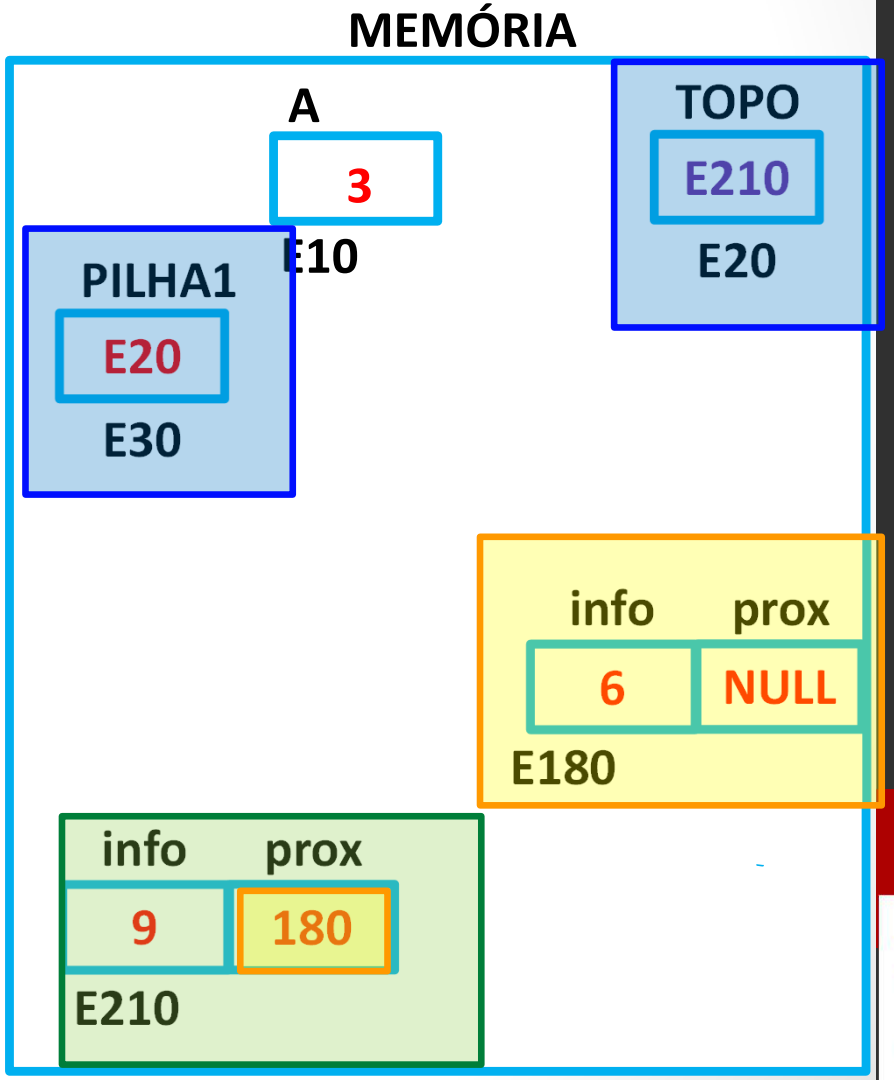
```
float pop (Pilha* p)
{
   float v;
   if (vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1); /* aborta programa
      */
   }
   v = p->TOPO->info;
   p->TOPO = ret_ini(p->TOPO);
   return v;
}
```




ESTRUTURA
DE
DADOS

Pilhas


PILHA1
9
6
//


MEMÓRIA
A
3
E10
TOPO
E210
E20
PILHA1
E20
E30
info
prox
6
NULL
E180
info
prox
9
180
E210

- **Exercícios**

Dada as seguintes Estruturas e Funções / Procedimentos

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

```
typedef struct pilha
{
        struct no *Topo;
} Pilha;
```

FUNÇÕES / PROCEDIMENTOS:

```
Pilha * cria();
push(Pilha *, int);
int pop(Pilha *);
```

Pede-se:

- Elabore um procedimento / função que inverta uma pilha
- Elabore um procedimento / função que conte quantos números pares possui uma pilha e retorne esta quantidade
- Elabore um procedimento / função que apague a informação 15 de uma pilha se ela existir