TAD Pilha

Prof. André Grégio

Tipos Abstratos de Dados

Relembrando…

- **Representação** de itens/objetos/elementos
- Possui **atributos** que abstraem as características dos itens representados
- Define **operações** que podem ser feitas sobre os itens

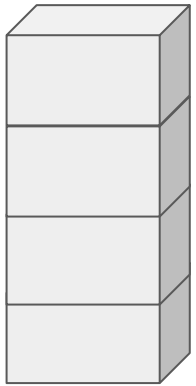
Tipos Abstratos de Dados

Operações comuns:

1. Inicializar TAD
2. Verificar se TAD está vazio
3. Criar elemento
4. Inserir elemento
5. Remover elemento
6. Buscar elemento

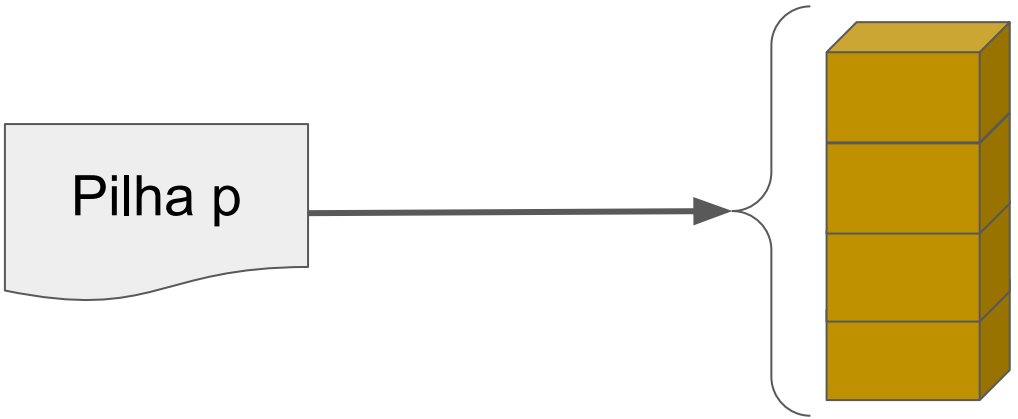
Pilha

A pilha é um tipo abstrato de dados especial que representa um conjunto de objetos aos quais só se tem acesso ao elemento do TOPO



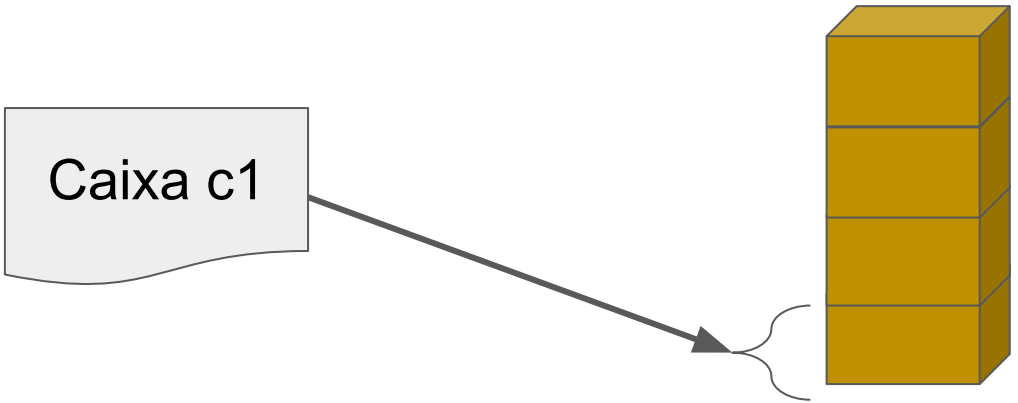
Pilha

Uma pilha de exemplo com 4 “caixas”



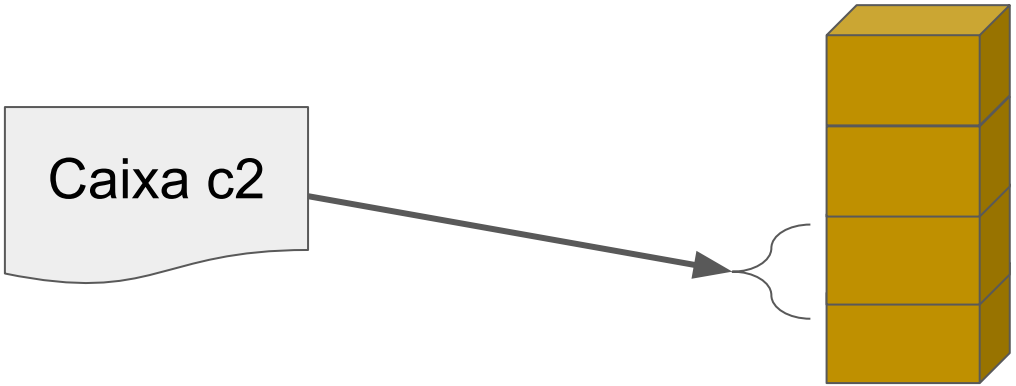
Pilha

Uma pilha de exemplo com 4 “caixas”



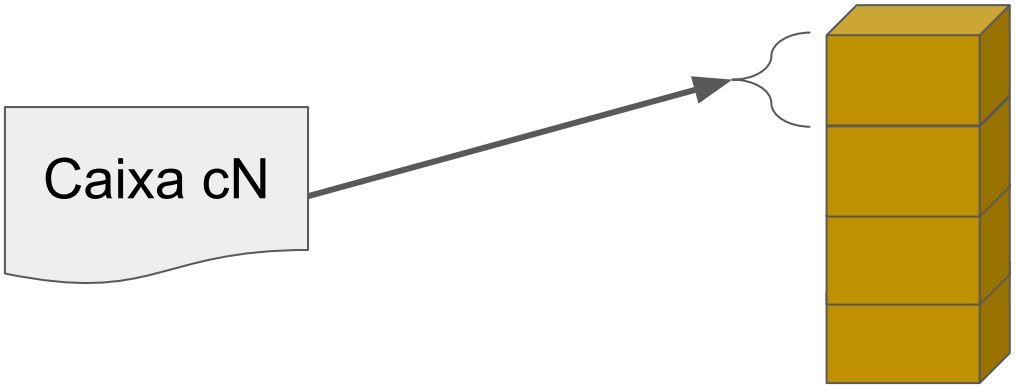
Pilha

Uma pilha de exemplo com 4 “caixas”



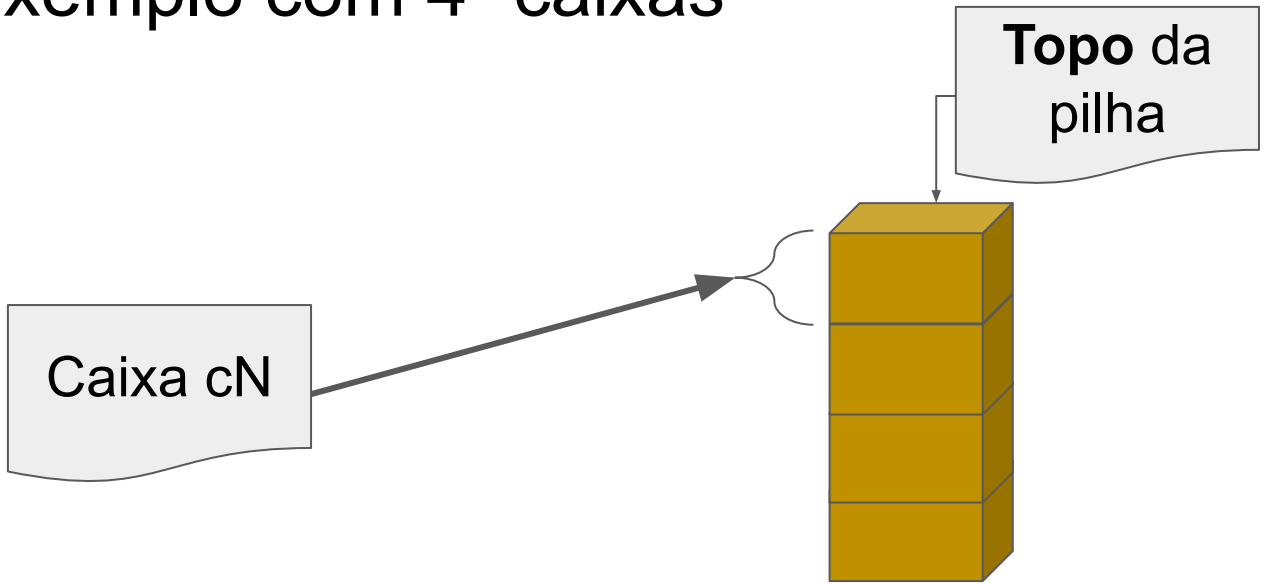
Pilha

Uma pilha de exemplo com 4 “caixas”



Pilha

Uma pilha de exemplo com 4 “caixas”



Estrutura da pilha

Uma pilha pode possuir os seguintes atributos:

- Topo, para marcação do elemento “acessível”
- Tamanho, para indicar quantos elementos estão na pilha
- Comprimento, para demarcar quantos elementos a pilha suporta
- Espaço de armazenamento, para guardar os elementos

Estrutura da pilha

Uma pilha pode possuir os seguintes atributos:

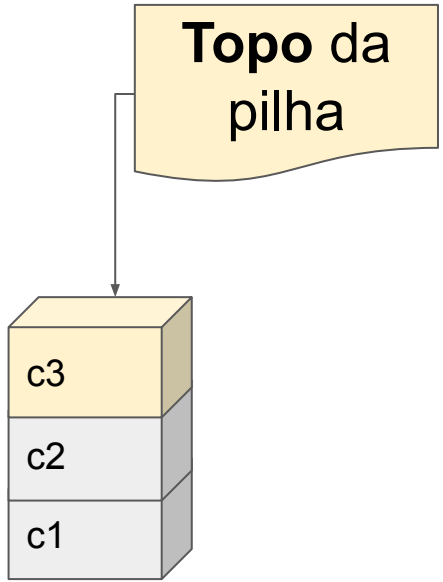
- Topo, para marcação do elemento “acessível”
- Tamanho, para indicar quantos elementos estão na pilha
- Comprimento, para demarcar quantos elementos a pilha suporta
- Espaço de armazenamento, para guardar os elementos

Nem todos os atributos listados acima são necessários!

Estrutura da pilha

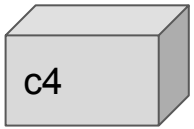
Suponha a seguinte pilha:

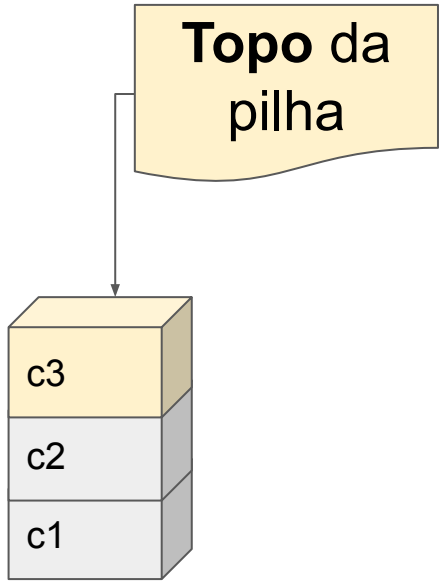
- Já inicializada com elementos do tipo caixa
  - Elemento “caixa” armazena um inteiro com seu valor
- Tamanho = 3 (possui três elementos)
- Comprimento = 4 (guarda até quatro elementos)
- Topo aponta para elemento “c3”



Operações sobre a pilha

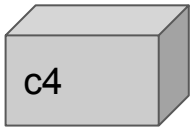
- Inserir elemento
  - Cria um novo elemento do tipo “caixa”



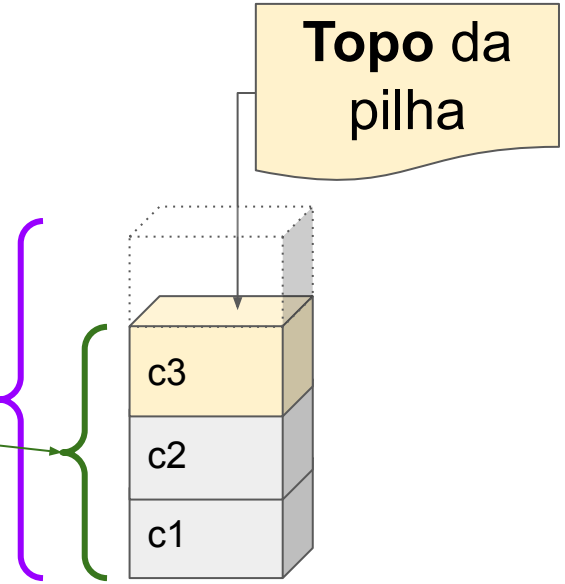


Operações sobre a pilha

- Inserir elemento
  - Cria um novo elemento do tipo “caixa”

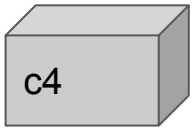


  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 3

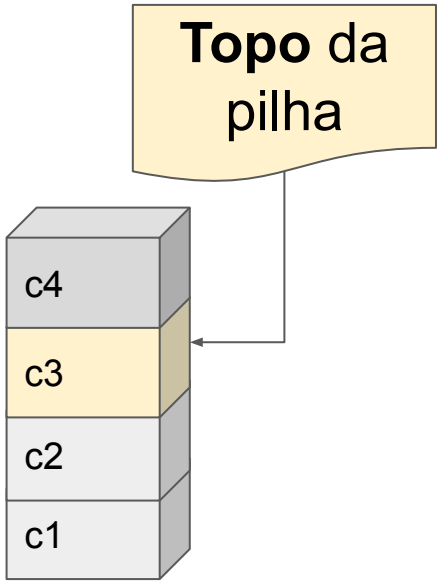


Operações sobre a pilha

- Inserir elemento
  - Cria um novo elemento do tipo “caixa”

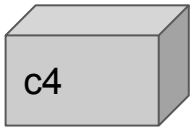


  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 3
  - Insere elemento na pilha

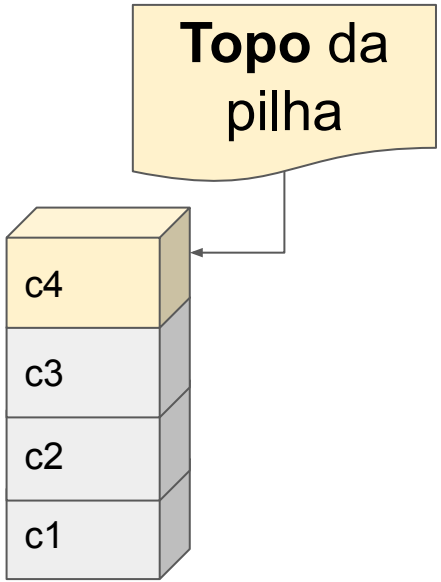


Operações sobre a pilha

- Inserir elemento
  - Cria um novo elemento do tipo “caixa”



  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 3
  - Insere elemento na pilha
  - Atualiza topo



Operações sobre a pilha

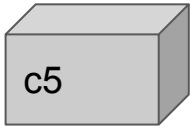
- A operação de **inserir elemento** em uma pilha é chamada de **PUSH**
- Argumentos de Entrada:
  - Uma pilha p
  - Um elemento x a ser inserido na pilha

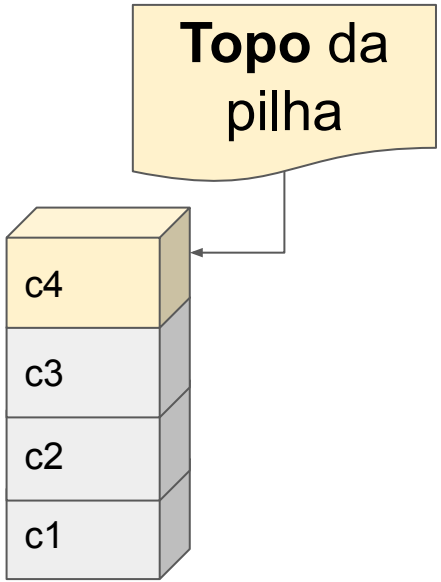
Protótipo da função PUSH:

- Push(pilha p, elemento x)

Operações sobre a pilha

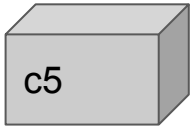
- Verificação de *overflow*
- Inserir elemento “c5”:
  - Cria um novo elemento do tipo “caixa”



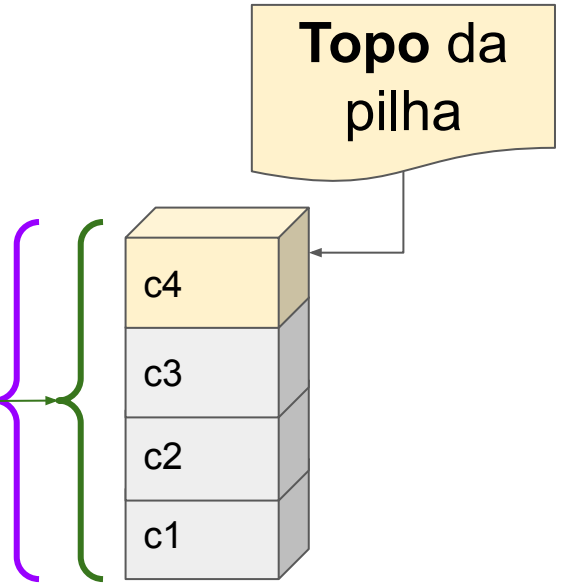


Operações sobre a pilha

- Verificação de *overflow*
- Inserir elemento “c5”:
  - Cria um novo elemento do tipo “caixa”

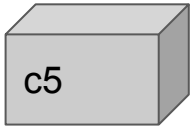


  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 4

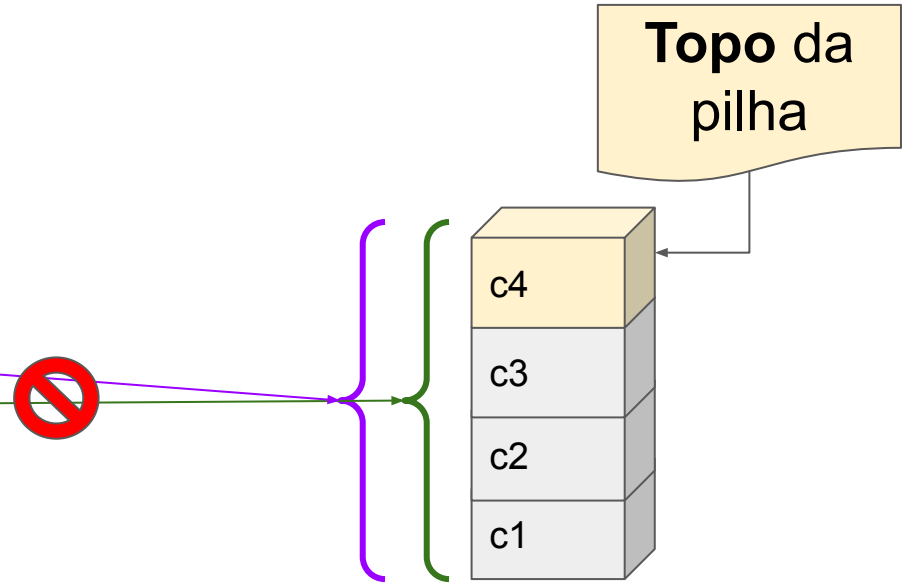


Operações sobre a pilha

- Verificação de *overflow*
- Inserir elemento “c5”:
  - Cria um novo elemento do tipo “caixa”

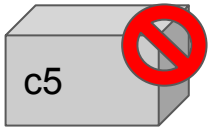


  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 4
  - Destrói novo elemento e avisa usuário

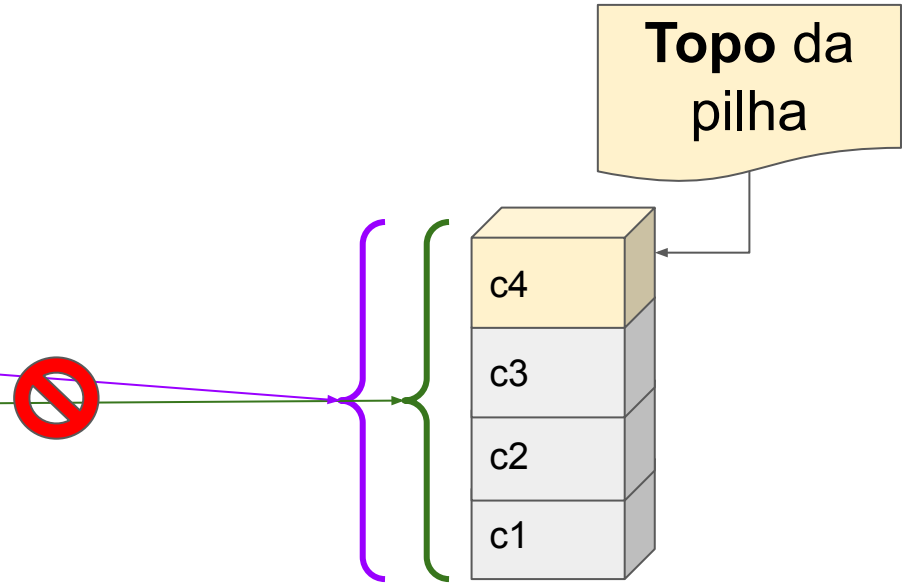


Operações sobre a pilha

- Verificação de *overflow*
- Inserir elemento “c5”:
  - Cria um novo elemento do tipo “caixa”

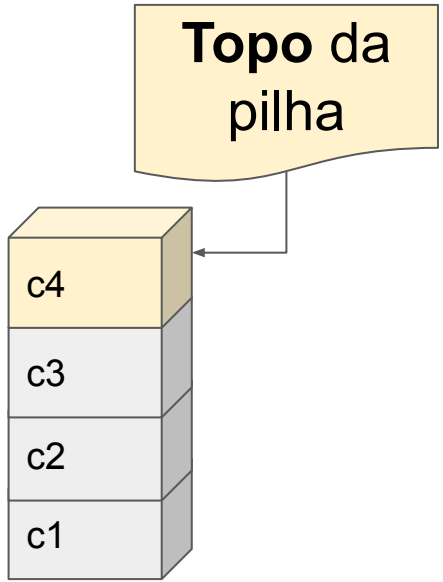


  - Verifica se há espaço na pilha
    - Comprimento = 4
    - Tamanho = 4
  - Destrói novo elemento e avisa usuário



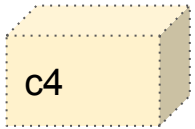
Operações sobre a pilha

- Remover elemento
  - Obtém o elemento do topo

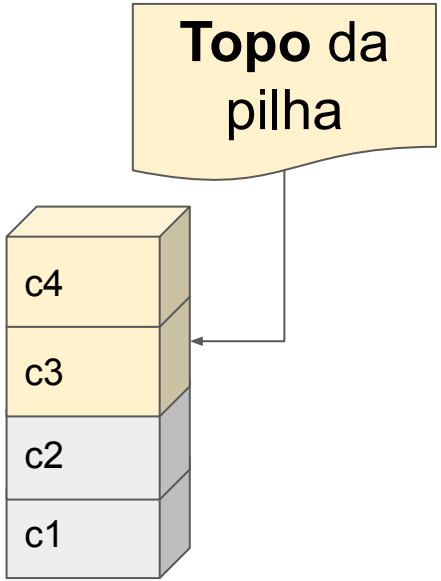


Operações sobre a pilha

- Remover elemento
  - Obtém o elemento do topo

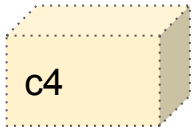


  - Atualiza topo

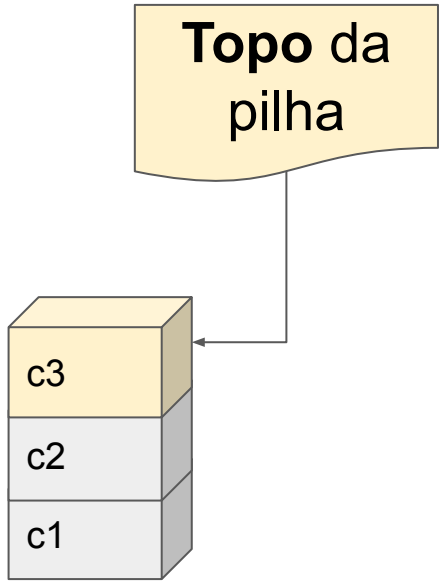


Operações sobre a pilha

- Remover elemento
  - Obtém o elemento do topo

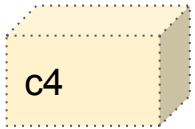


  - Atualiza topo
  - Libera memória (remove elemento do topo)



Operações sobre a pilha

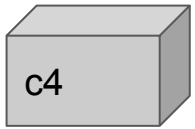
- Remover elemento
  - Obtém o elemento do topo

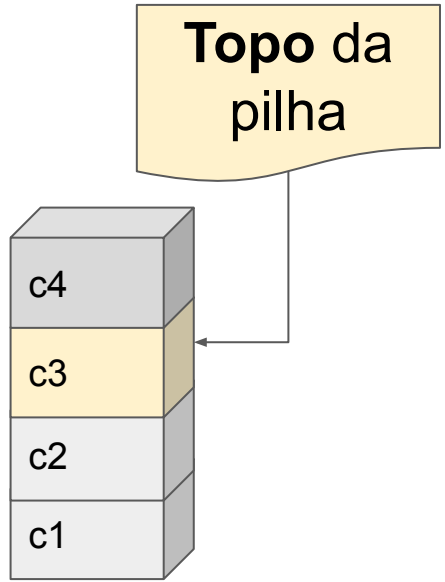


  - Atualiza topo
  - Libera memória (remove elemento do topo)

ou

  - Devolve elemento para quem chamou a função





Operações sobre a pilha

- A operação de **remover elemento** da pilha é chamada de **POP**
- Argumentos de Entrada:
  - Uma pilha p

Protótipo da função POP:

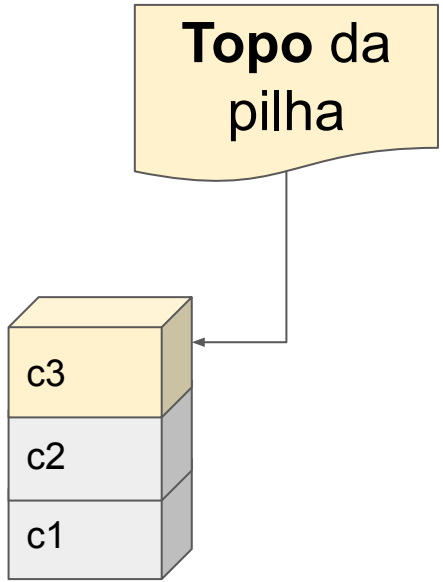
- Pop(pilha p)

Operações sobre a pilha

- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo

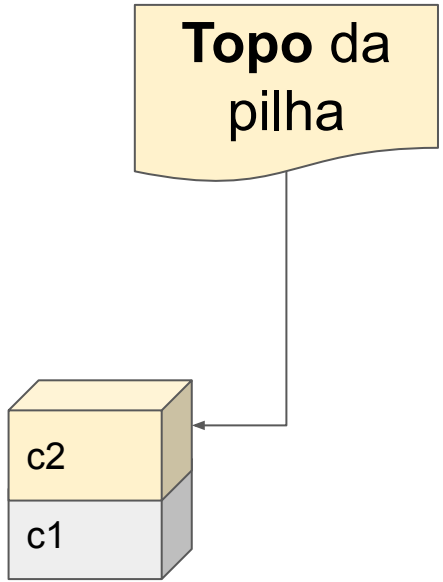
Operações sobre a pilha

- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo
- Pop(p)



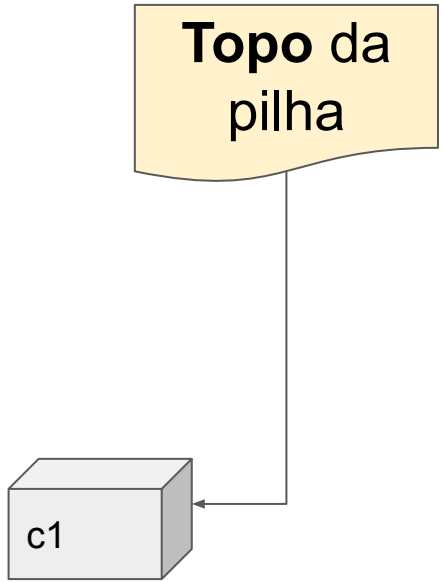
Operações sobre a pilha

- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo
- Pop(p)
- Pop(p)



Operações sobre a pilha

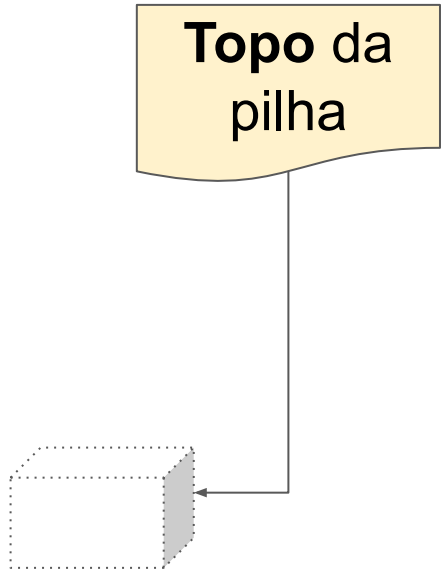
- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo
- Pop(p)
- Pop(p)
- Pop(p)



Operações sobre a pilha

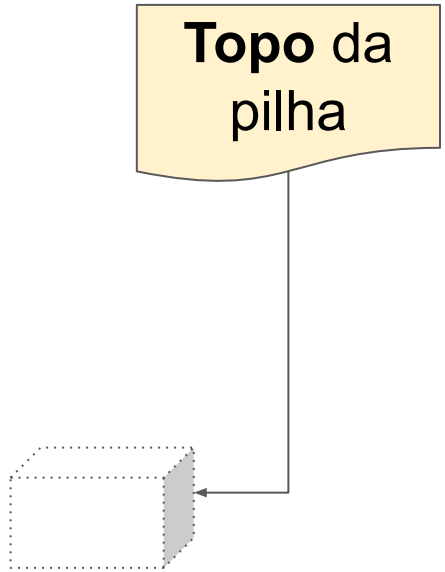
- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo
- Pop(p)
- Pop(p)
- Pop(p)

Pilha está vazia!



Operações sobre a pilha

- A operação de **remover elemento** da pilha (**POP**):
  - Não permite escolha do elemento, pois sempre remove o topo
- Pop(p)
- Pop(p)
- Pop(p)
- Pop(p)
- Tomar cuidado com *UNDERFLOW*!
  - Verificar se pilha está vazia...



Política da pilha

As pilhas obedecem à política LIFO (*Last In, First Out*), isto é:

- O último objeto a ser inserido é o primeiro objeto a ser removido
  - Insere no topo
  - Remove do topo

Exemplo de implementação do TAD pilha

Estrutura “item” com atributos “valor” e “próximo item”:

```
struct item {
        int valor;
        struct item *prox;
};
```

```
struct item *c1;
c1 = malloc(sizeof(struct item));
c1->valor = 1;
```

Exemplo de implementação do TAD pilha

Estrutura “pilha” com atributo “topo”

```
struct pilha {
        struct item *topo;
        int tamanho;
};
```

```
/* Declaração de variável p1 do tipo ponteiro para struct pilha
*/
struct pilha *p1;
```

Exemplo de implementação do TAD pilha

Operações sobre o TAD pilha:

- Inicializar

```
struct pilha *inicializaPilha(){
        struct pilha *p1;
        p1 = malloc(sizeof(struct pilha));
        p1->topo = NULL;
        p1->tamanho = 0;
        return p1;
}
```

```
struct pilha *p1 = inicializaPilha();
```

Exemplo de implementação do TAD pilha

Operações sobre o TAD pilha:

- Verificar se está vazia:

```
int pilhaVazia(struct pilha *p1){
    if (p1) {
        if !(p1->topo)
            /* pilha vazia */
            return 1;
        /* pilha não-vazia */
        return 0;
    }
    /* pilha não-inicializada */
    return -1;
}
```

Exemplo de implementação do TAD pilha

Operações sobre o TAD pilha:

- Criar elemento:

```
struct item *criaItem(int valor){
        struct item *tmp;
        tmp = malloc(sizeof(struct item));
        tmp->valor = valor;
        tmp->prox = NULL;
        return tmp;
}
```

Exercício

Implementar o TAD pilha em forma de biblioteca (.h e .c), com todas as operações dadas, inclusive:

- PUSH
- POP (com retorno de item e liberação de memória na função principal)
- Mostra topo
- Mostra quantos elementos tem na pilha

ATENÇÃO: verificar por *overflow* e *underflow*!