Tipo Abstrato de Dados:
Pilhas

Rafael Viana de Carvalho

Estrutura de Dados

Tipos Abstratos de Dados (TAD)

- Abstraída qualquer linguagem de programação, um **TAD** pode ser visto como um modelo matemático que encapsula um **modelo de dados** e um conjunto de **procedimentos** que atuam com exclusividade sobre os dados encapsulados
  - Qualquer processamento a ser realizada sobre os dados encapsulado só poderá ser executada por intermédio dos procedimentos definidos no modelo matemático
  - Existem quatro operações básicas de processamento:
    - Criação
    - Inclusão
    - Remoção
    - Percurso (busca)

Tipos Abstratos de Dados

- Existem basicamente dois tipos de **Estruturas de Dados** que implementam um TAD:
  - Estruturas lineares - mantém os itens de informação de forma independente de seus valores.
    - A única informação utilizada pela estrutura é a posição do item
    - Qualquer manipulação relativa ao conteúdo ou valor desse item é atribuição da aplicação.
    - Exemplo: Pilhas, Filas e Listas
  - Estruturas não lineares (associativas) - permitem o acesso a seus elementos de forma independente de sua posição, com base no seu valor
    - Exemplos: Árvores e Grafos

Pilha (STACK)

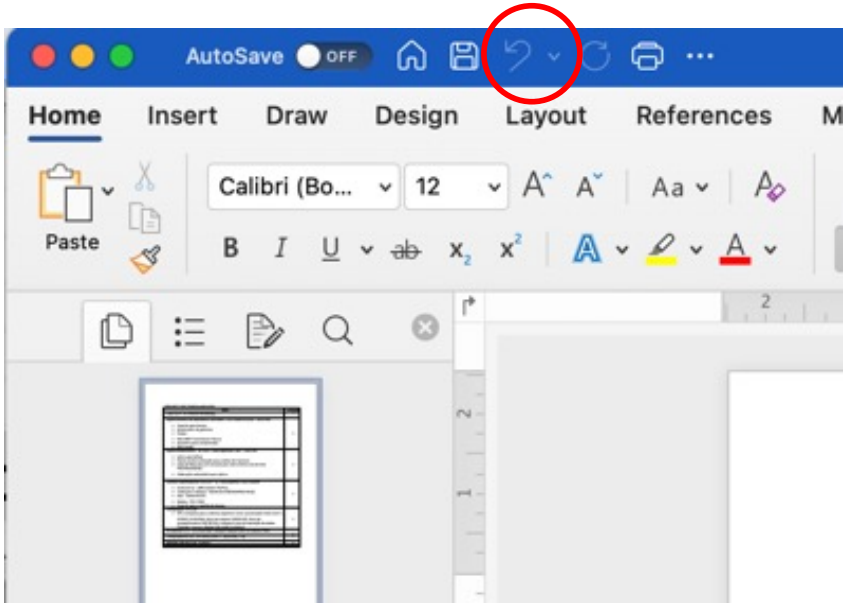
- Em computação:
  - Quando ouvir o termo “**pilha**”, pense primeiro nos pratos!!!
- É a estrutura de dados mais utilizada em programação
  - **Usos**: Chamada de subprogramas, avalição de expressões aritméticas, etc.

Pilha (STACK)

- A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo
  - Quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo
    - O único elemento que pode ser removido da pilha é o do topo
  - Os elementos da pilha são retirados na ordem inversa à ordem em que foram introduzidos
    - O primeiro que sai é o último que entrou (LIFO – last in, first out)

Pilha (STACK)

- Exemplos:
  - Já observou o recurso de “desfazer” do Word?



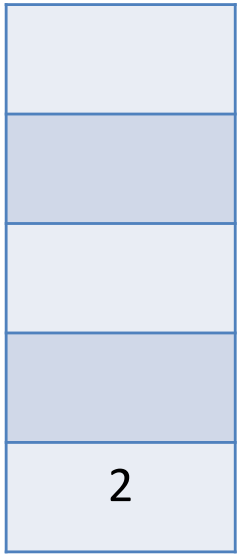
  - Qual operação ele desfaz?
  - Word coloca as operações em uma pilha!

Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Empilhar!

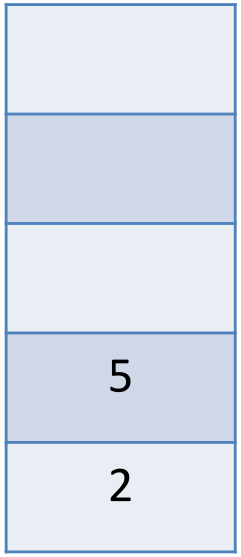


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Empilhar!

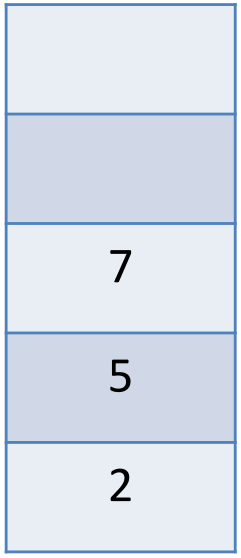


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Empilhar!

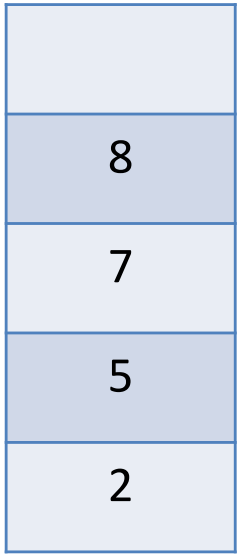


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Empilhar!



Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Empilhar!

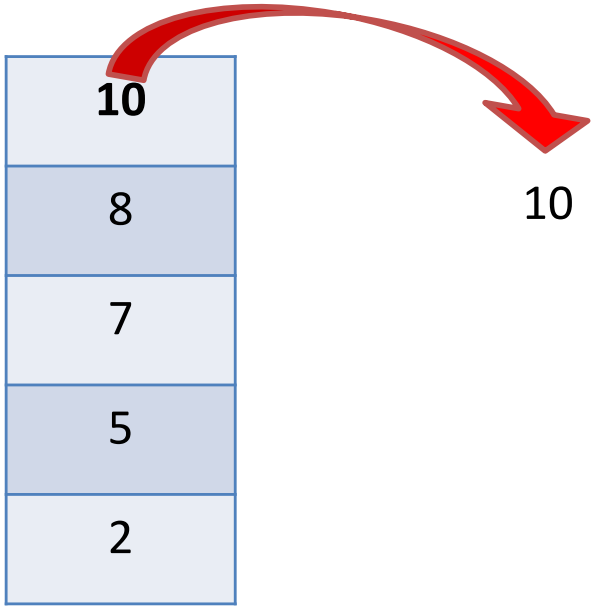
10
8
7
5
2

Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Desempilhar!

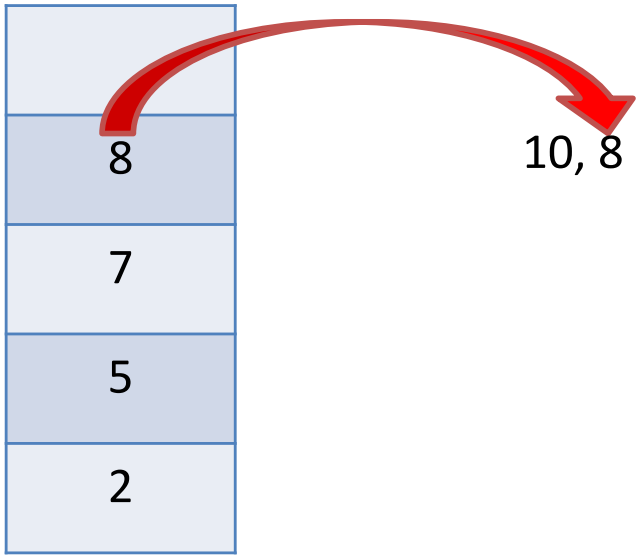


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Desempilhar!

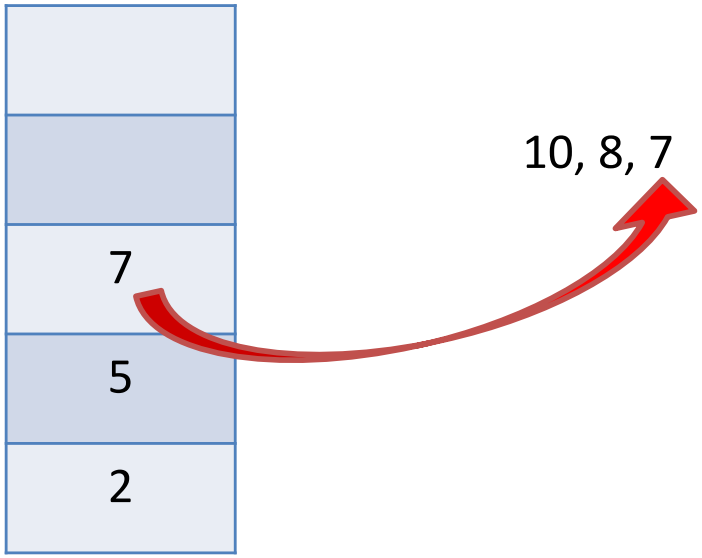


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Desempilhar!

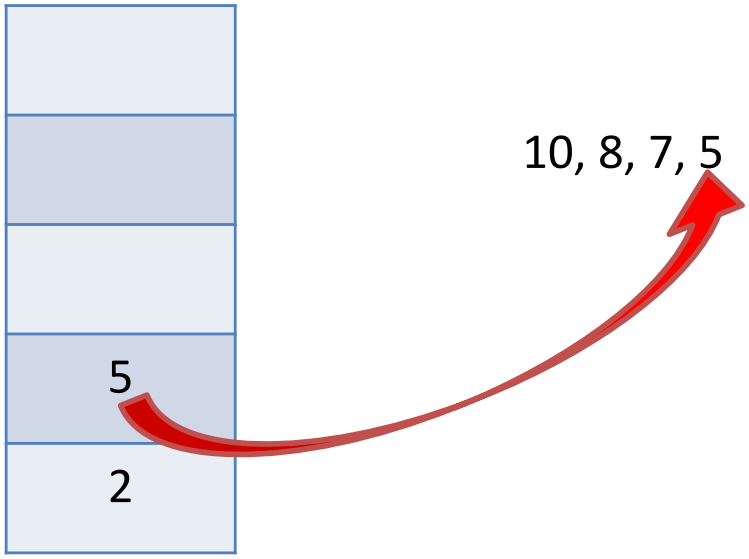


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

2	5	7	8	10

- Uma série de trocas... Ou...
- Desempilhar!

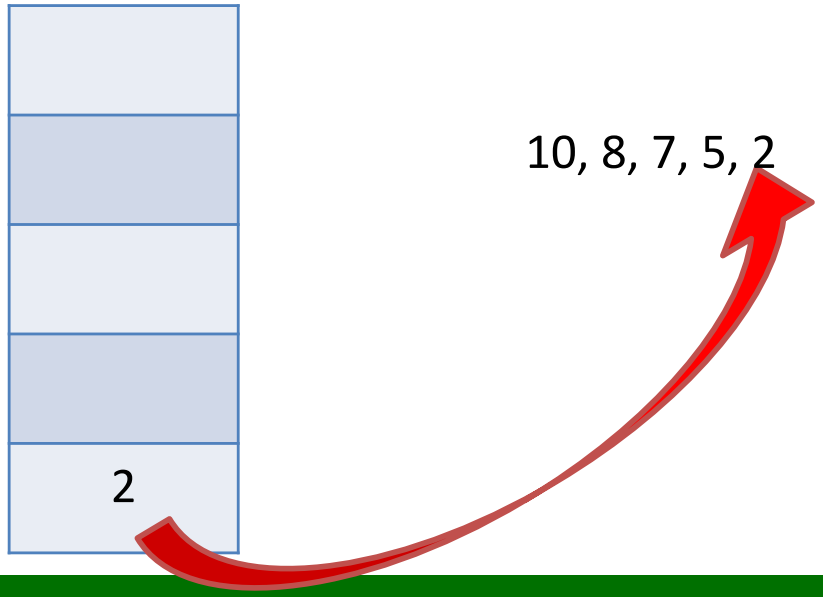


Pilha (STACK)

- Exemplos:
- Se você tem uma lista crescente...

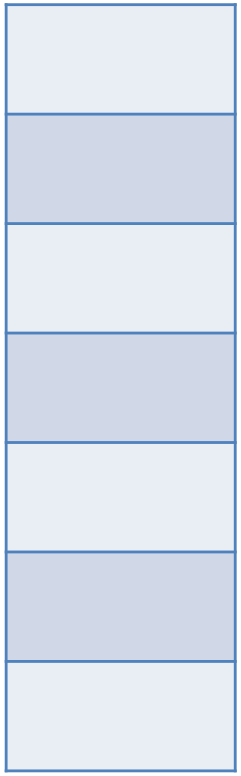
2	5	7	8	10

- Uma série de trocas... Ou...
- Desempilhar!



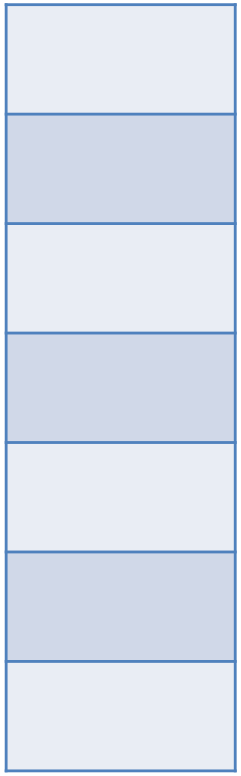
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



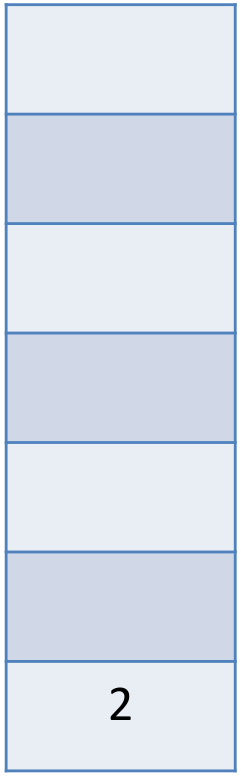
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



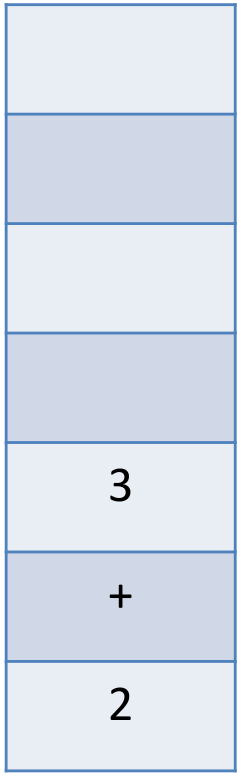
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilha (STACK)

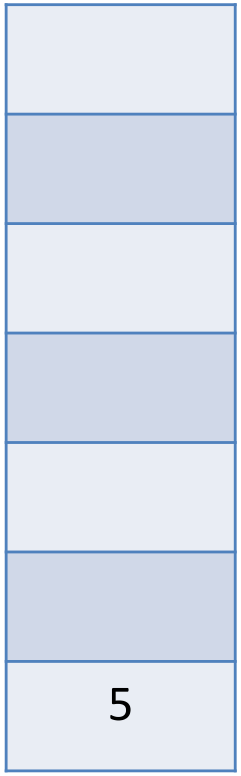
- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



3 + 2 = 5

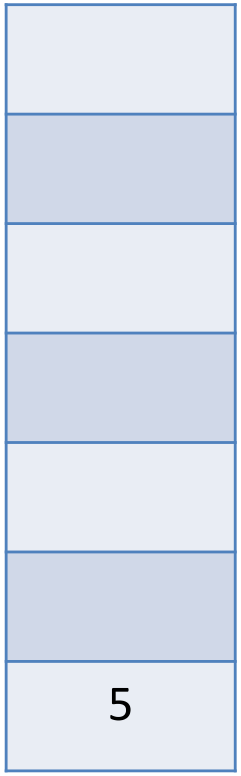
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



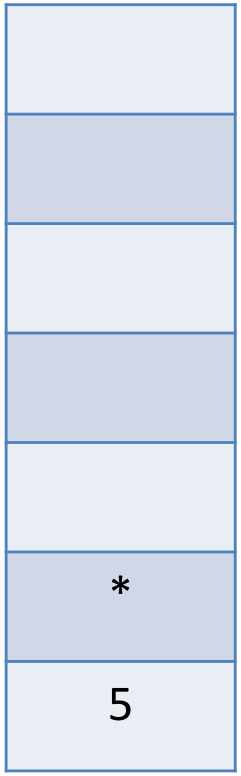
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



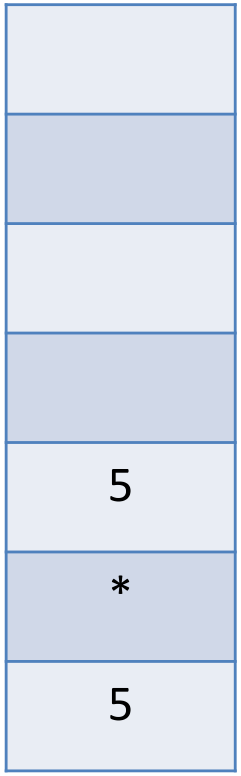
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilha (STACK)

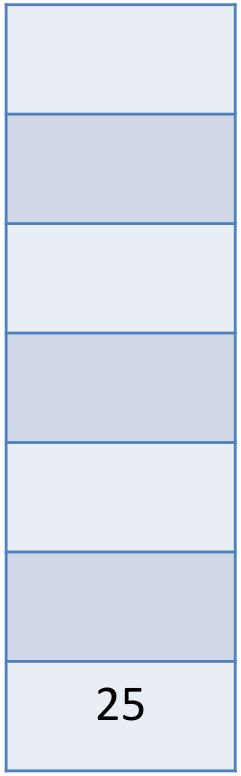
- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



5 * 5 = 25

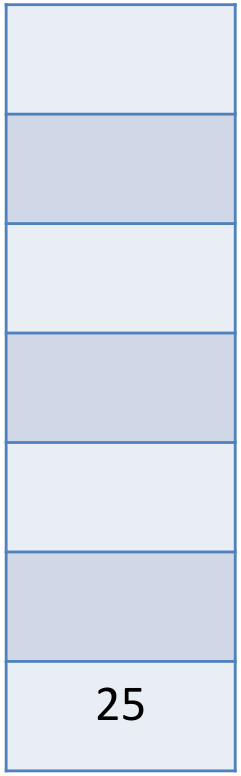
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



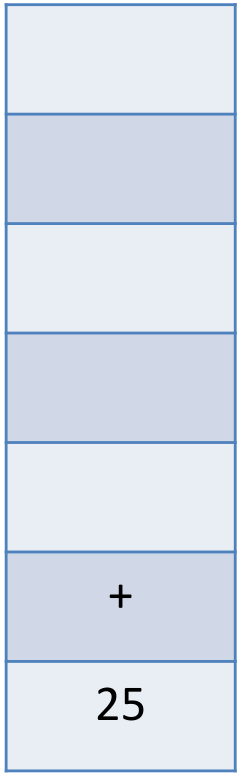
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



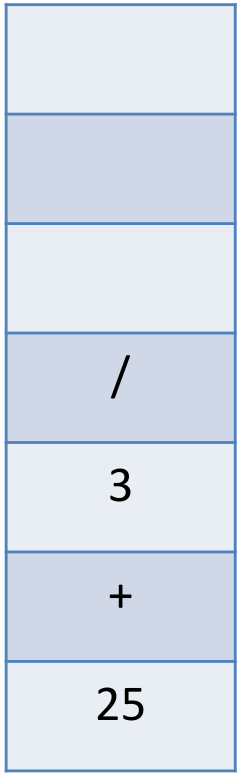
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



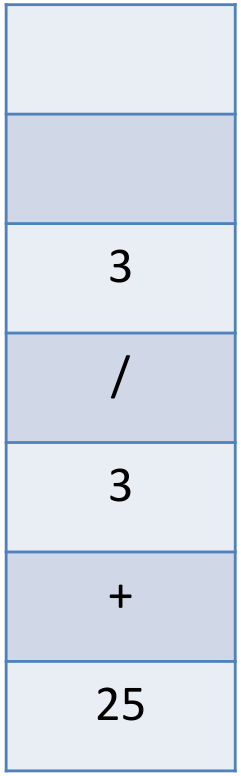
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



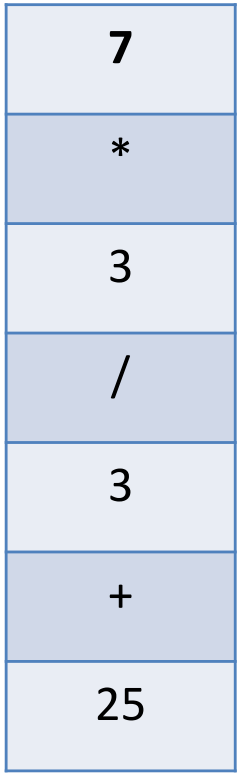
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilha (STACK)

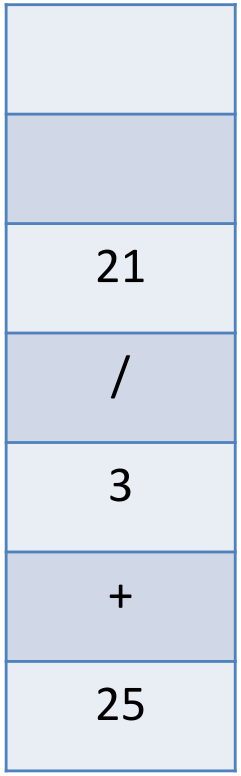
- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



7 * 3 = 21

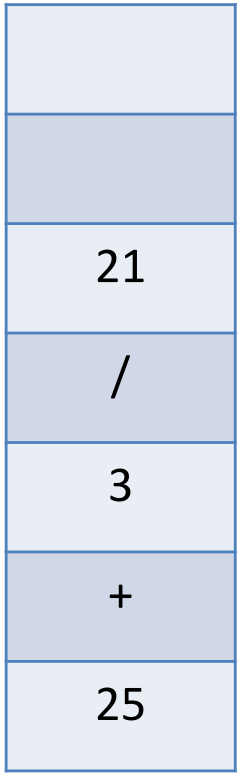
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilha (STACK)

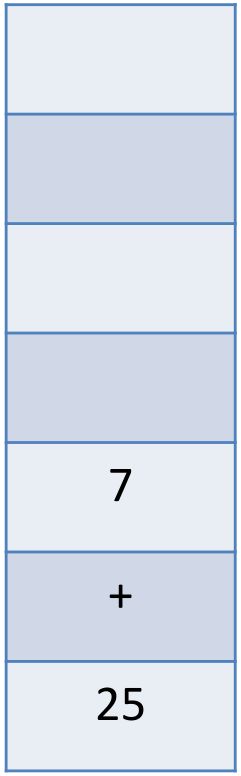
- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



21 / 3 = 7

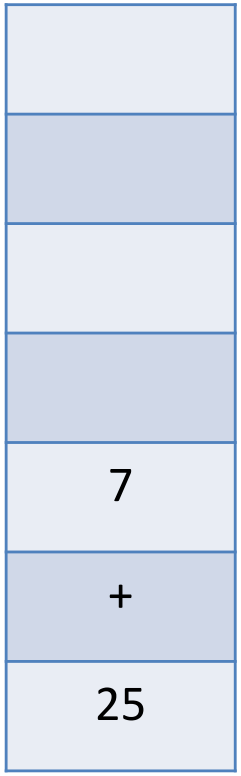
Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilha (STACK)

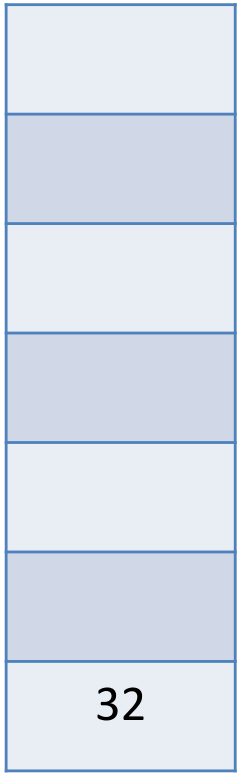
- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



7 + 25 = 32

Pilha (STACK)

- Exemplos:
- Como fazemos esse cálculo?
  (((2 + 3) * 5) + (3 / (3 * 7)))



Pilhas

- São estruturas formadas por um conjunto ordenado de dados no qual a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo.
- Operações associadas:
  -
  -
  -
  -
- Ultimo elemento a entrar é o primeiro a sair
- Aplicação: guardar variáveis locais em chamas recursirvas

Empilha B

Pilha Vazia

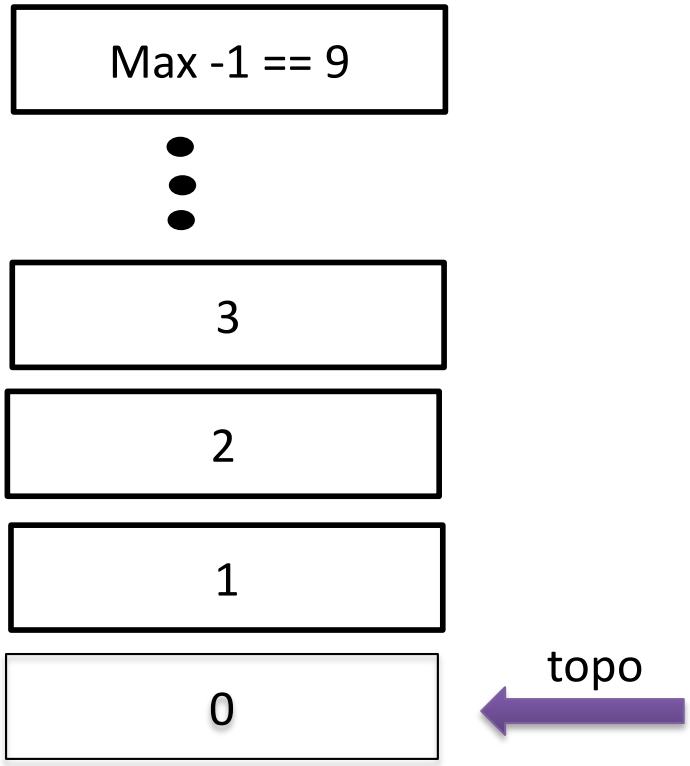
Implementando uma pilha

- Armazenamento em posições contínuas em um vetor (array)
- É necessário apenas saber a posição do elemento que está no topo
  - Usa-se um inteiro para armazenar seu índice no array.

```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
```

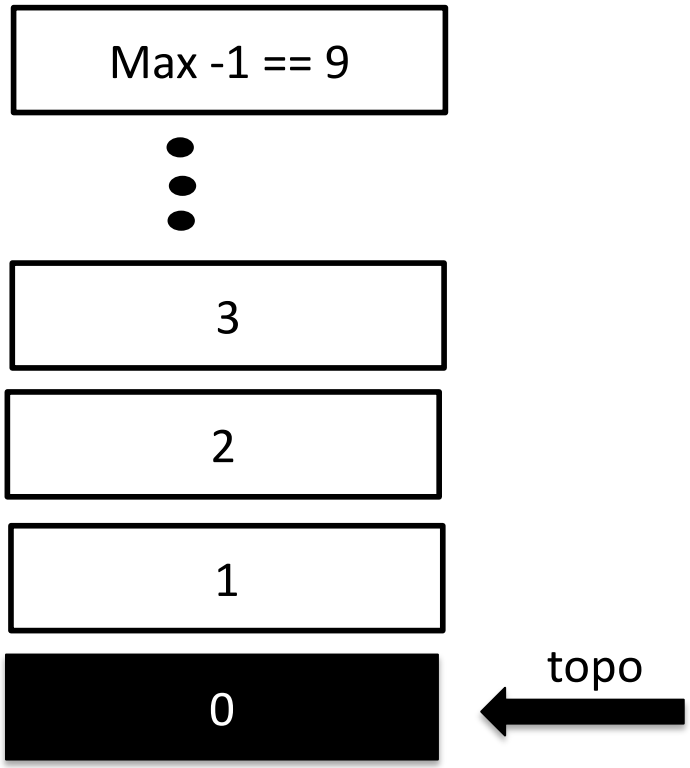
Criando uma pilha




```
#define Max 10

struct pilha {
        int elementos[MAX];
        int topo;
}
struct pilha * cria(void) {
        struct pilha *p;
        p = malloc(sizeof(struct pilha));
        if(!p) {
                perror(NULL);
                exit(1);
        }
        /* IMPORTANTE: */
        p->topo = 0;
}
```

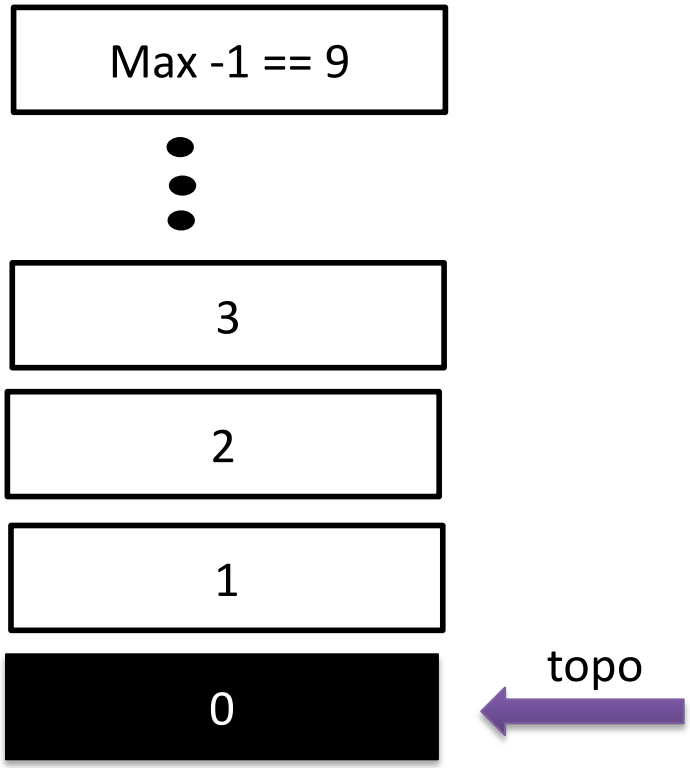
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 7123);
 ..
}
```

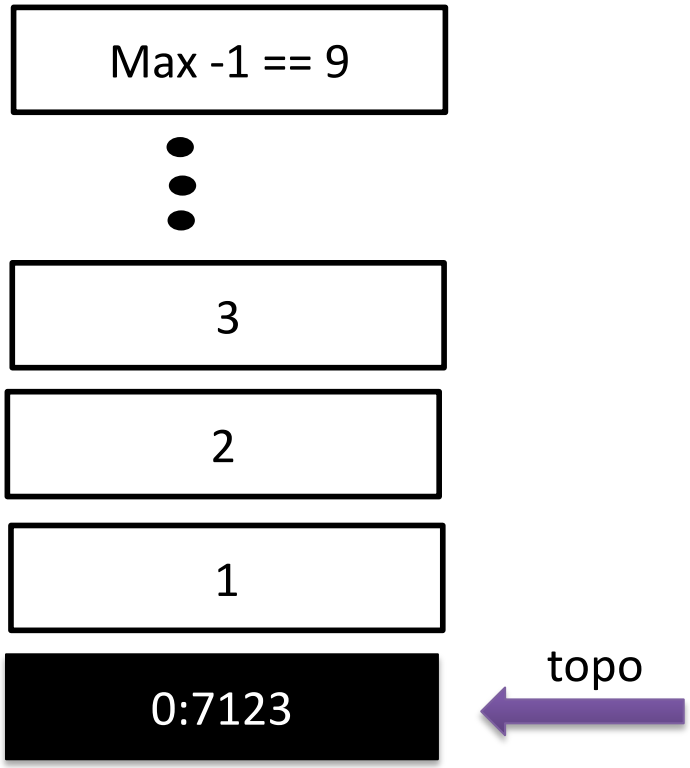
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 7123);
...
}
```

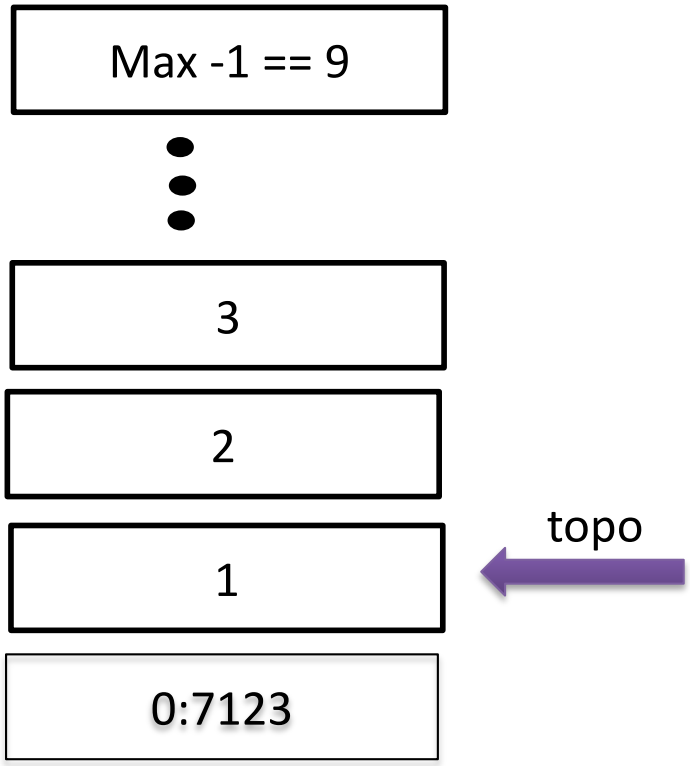
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 7123);
...
}
```

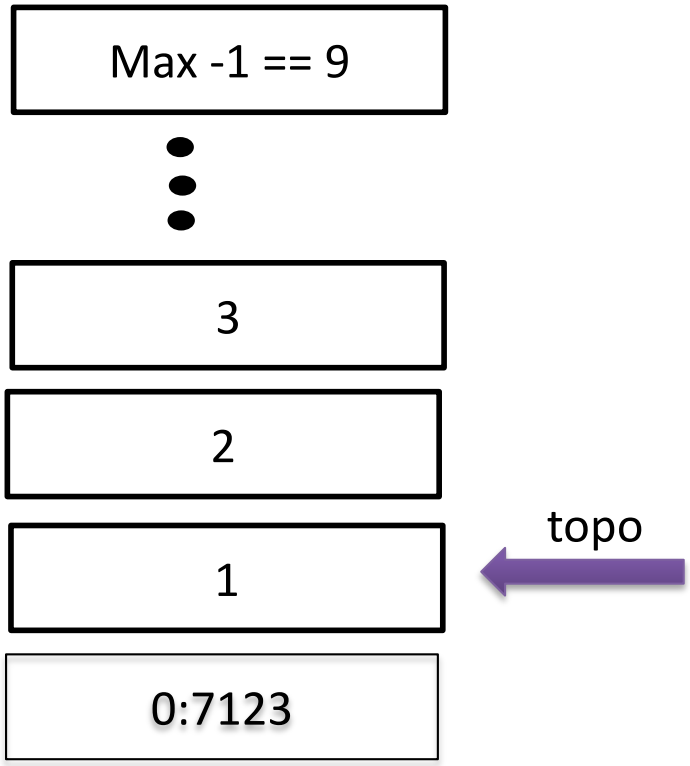
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 7123);
...
}
```

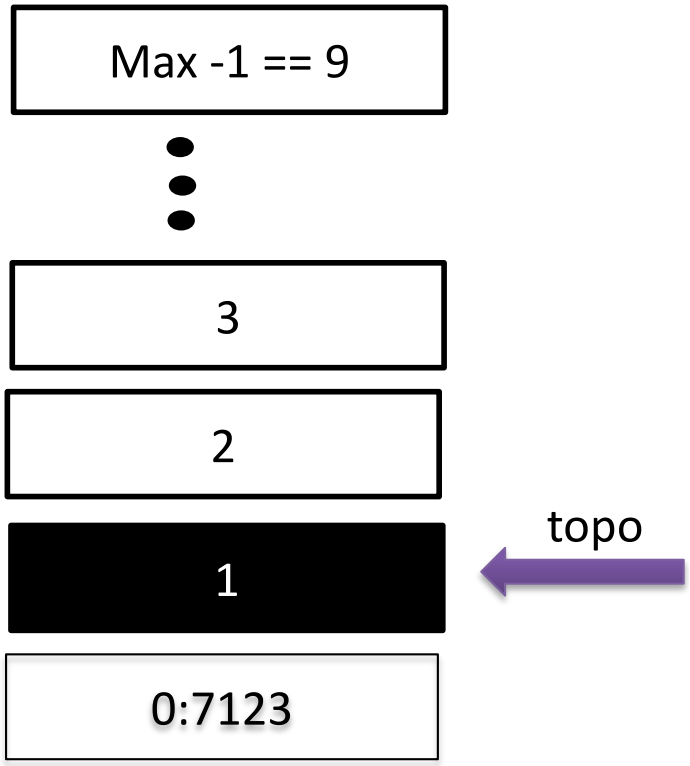
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 1201);
 ..
}
```

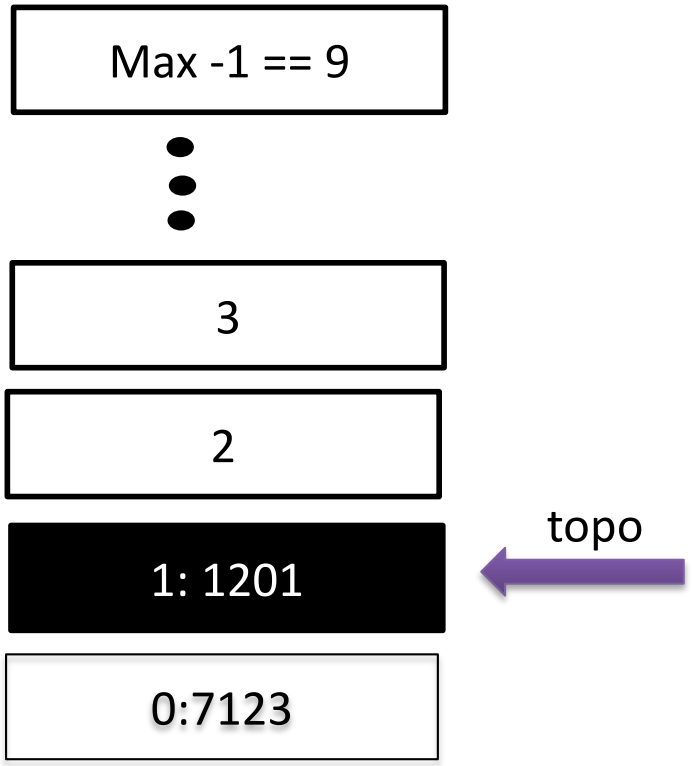
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 1201);
...
}
```

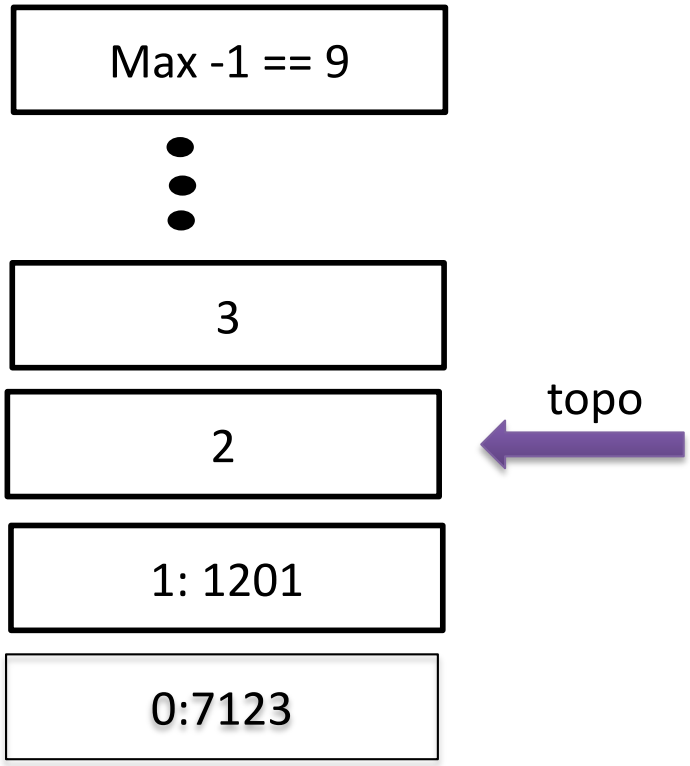
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 1201);
...
}
```

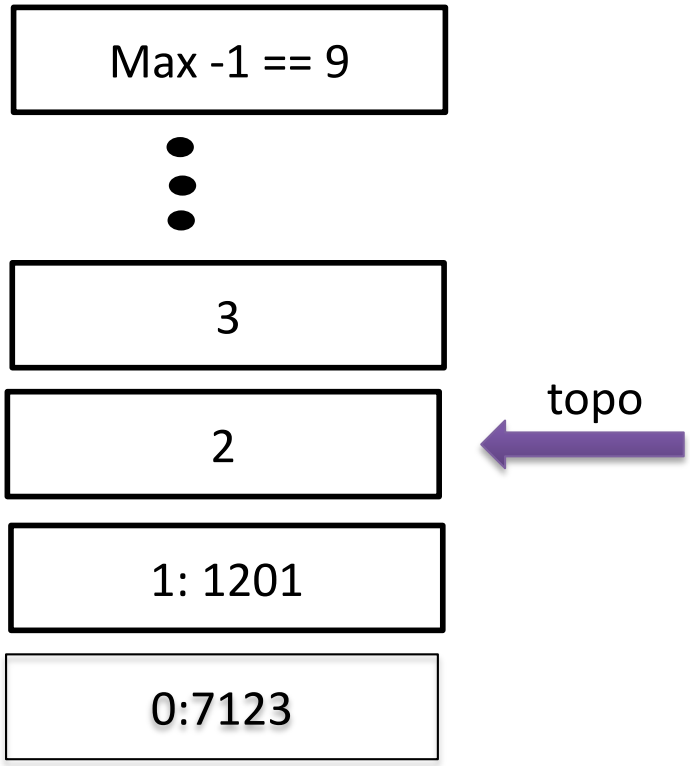
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 1201);
...
}
```

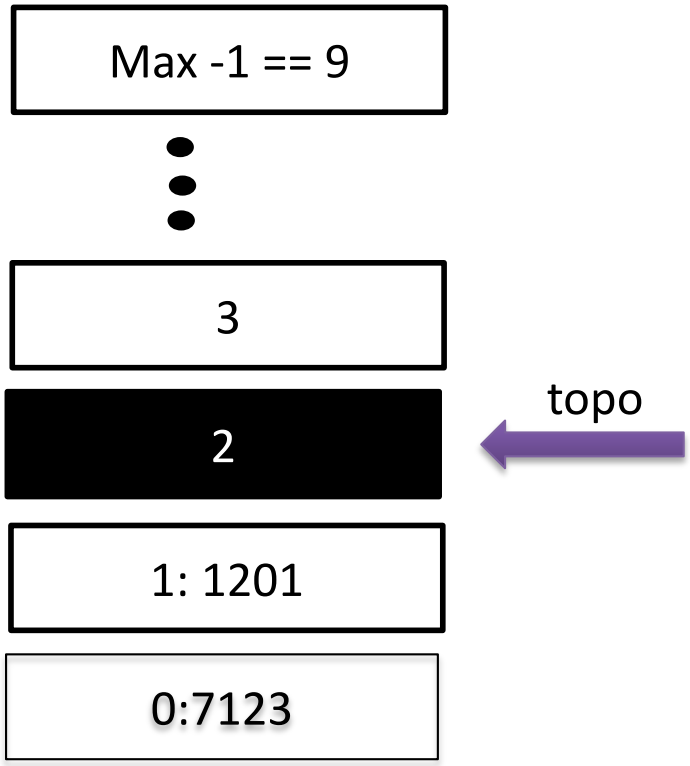
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 8905);
...
}
```

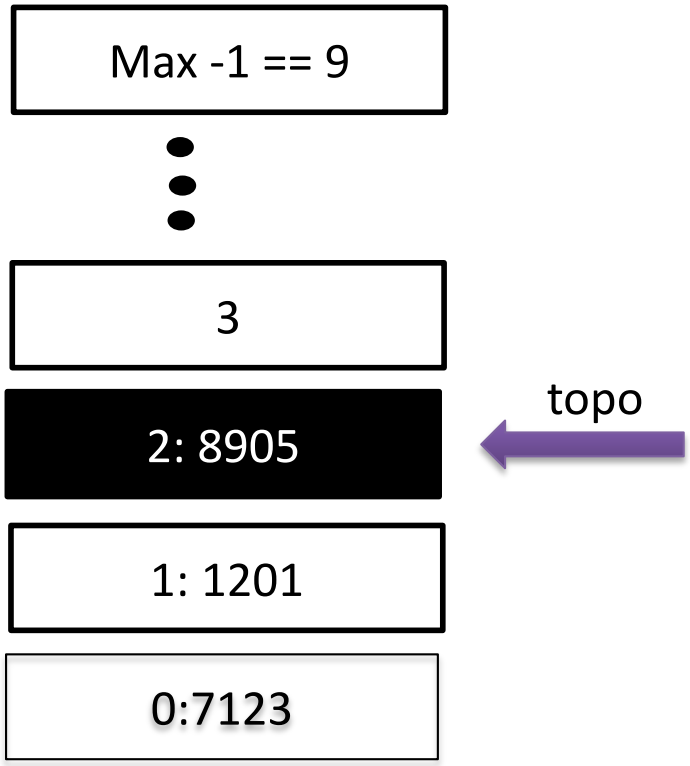
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 8905);
...
}
```

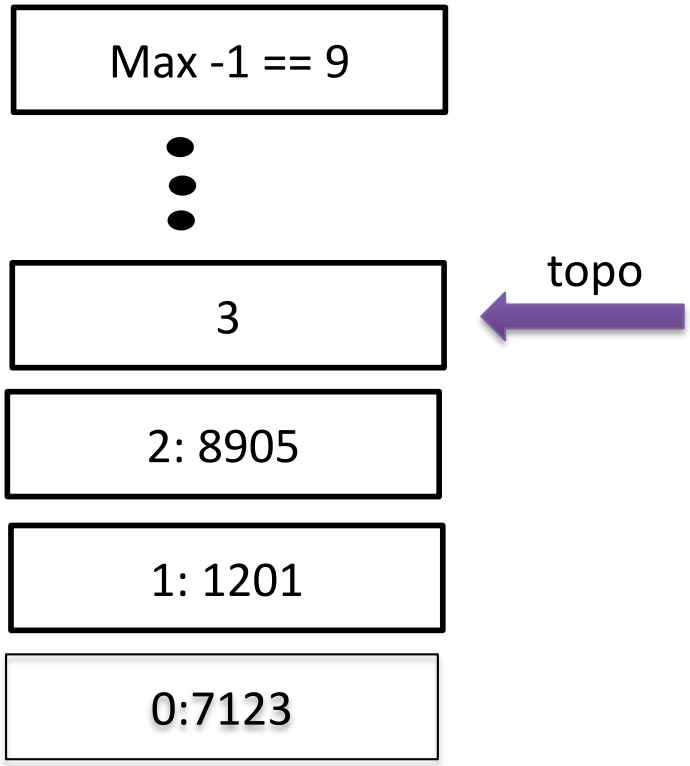
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 8905);
...
}
```

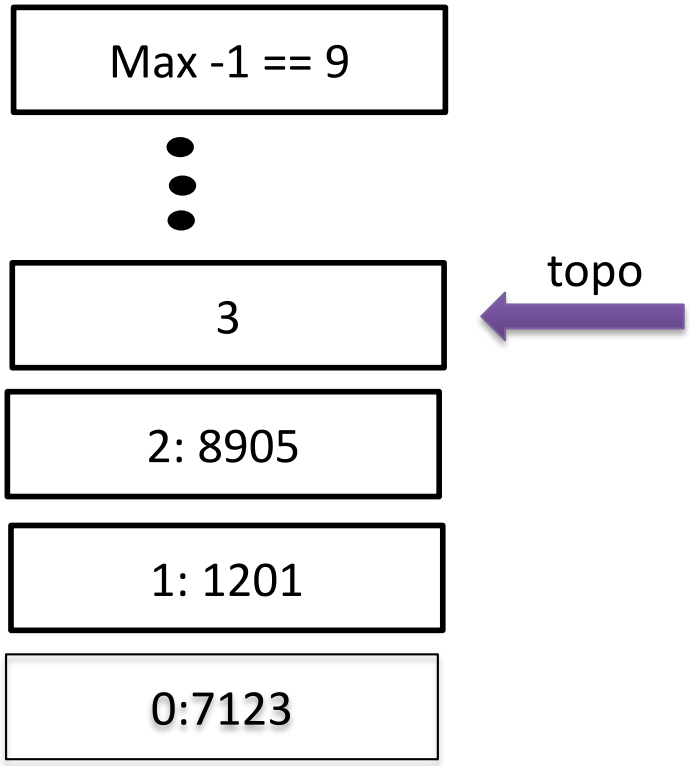
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 8905);
...
}
```

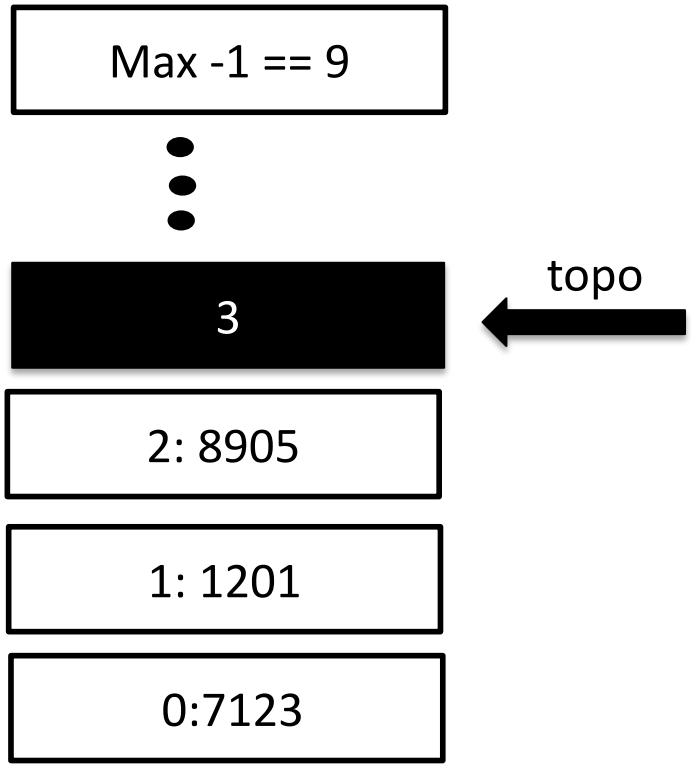
Empilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
Void empilha (struct pilha *p, int Numero) {
        p->elementos[p->topo] = Numero;
        p ->topo = p->topo +1;
}
Void main () {
...
empilha (p, 8905);
...
}
```

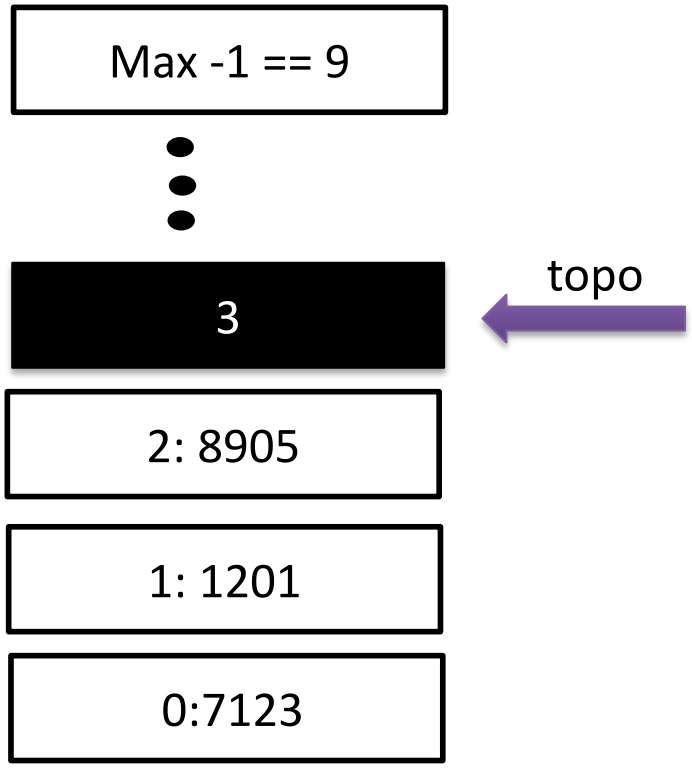
Desempilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
int desempilha (struct pilha *p) {
        p->topo = p->topo-1;
        return p->elementos[p->topo];
}
Void main () {
...
Int t = desempilha (p);
...
}
```

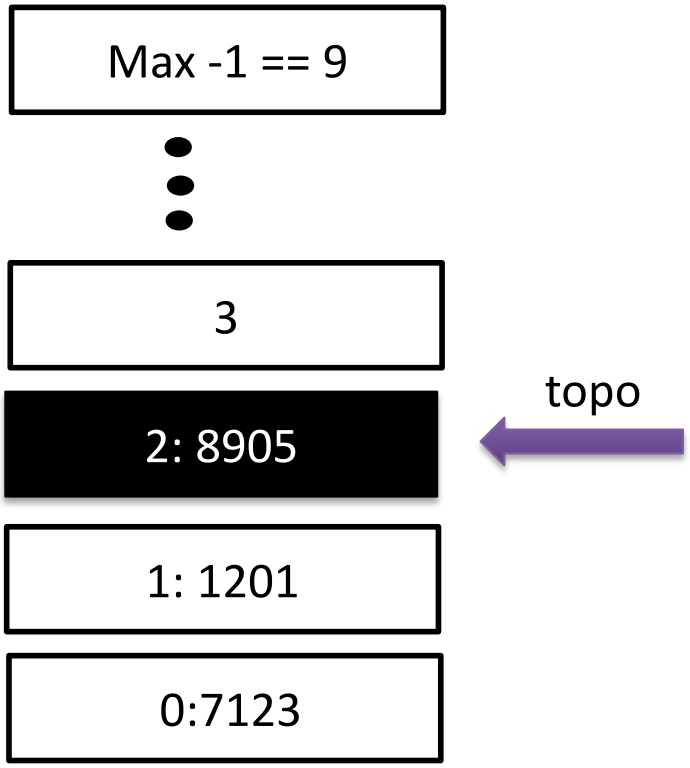
Desempilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
int desempilha (struct pilha *p) {
        p->topo = p->topo-1;
        return p->elementos[p->topo];
}
Void main () {
...
Int t = desempilha (p);
...
}
```

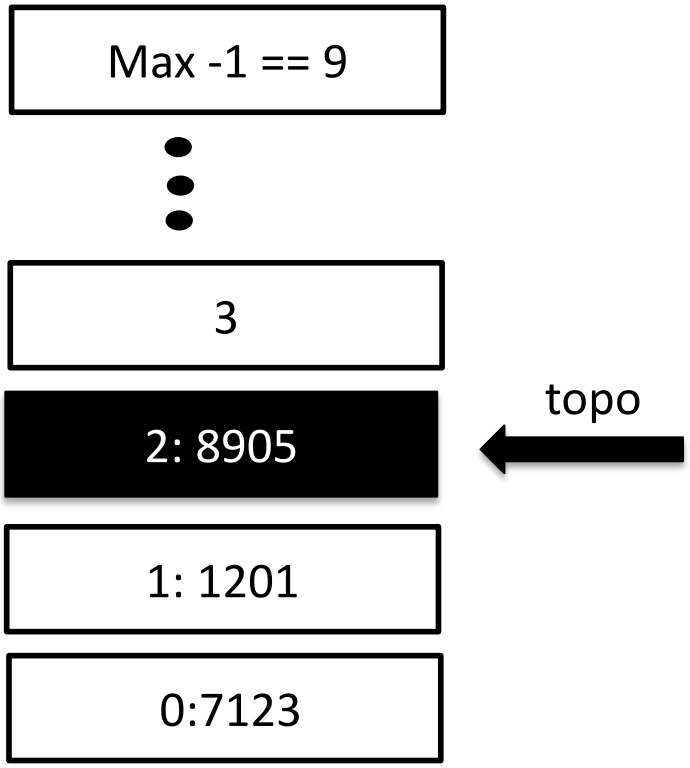
Desempilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
int desempilha (struct pilha *p) {
        p->topo = p->topo-1;
        return p->elementos[p->topo];
}
Void main () {
...
Int t = desempilha (p);
...
}
```

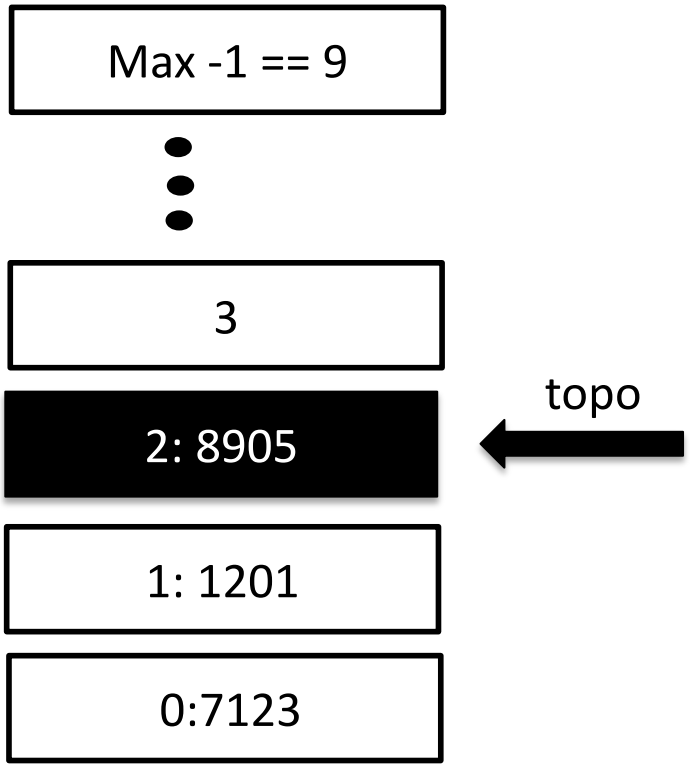
Desempilhando um elemento




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
int desempilha (struct pilha *p) {
        p->topo = p->topo-1;
        return p->elementos[p->topo];
}
Void main () {
...
Int t = desempilha (p);
/* t == 8905 */
...
}
```

Destruindo uma pilha




```
#define Max10

struct pilha {
        int elementos[MAX];
        int topo;
}
int destroi (struct pilha *p) {
        free(pilha);
}
Void main () {
...
destroi (p);
...
}
```

Empilhando e Desempilhando

- É importante testar se a pilha está cheia (antes de empilhar um elemento) ou vazia antes de desempilhar um elemento

```
Void empilha (struct pilha *p, int Numero) {
        if(p->topo == MAX-1) {
                printf("pilha cheia.");
                exit(1);
        }
        p->elementos[p->topo] = A;
        p ->topo = p->topo +1;
}
```

```
int desempilha (struct pilha *p) {
        if(p->topo == 0) {
                printf("pilha vazia.");
                exit(1);
        }
        p->topo = p->topo-1;
        return p->elementos[p->topo];
}
```

Operações Básicas

- 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 Básica com um vetor

- Como seria as funções de
  - Criar Pilha
  - Empilhar (PUSH)
  - Desempilhar (POP)
  - Pilha Vazia

Implementação Básica com um vetor

```
#define MAX 50
struct pilha {
        int topo;
        float vet[MAX];
};
Pilha* cria (void) {
        Pilha* p = (Pilha*) malloc(sizeof(Pilha));
        p->topo = 0; /* inicializa com zero elementos */
        return p;
}
```

Implementação Básica com um vetor

```
void push (Pilha* p, float v) {
        if (p->topo == MAX) { /* capacidade esgotada */
                printf("Capacidade da pilha estourou.\n");
                exit(1); /* aborta programa */
        }
        /* insere elemento na próxima posição livre */
        p->vet[p->topo] = v;
        p->topo++;
}
```

Implementação Básica com um vetor

```
float pop (Pilha* p) {
        float v;
        if (p->topo==0) {
                printf("Pilha vazia.\n");
                exit(1); /* aborta programa */
        }
        /* retira elemento do topo */
        p->topo--;
        v = p->vet[p->topo];
        return v;
}
```

Implementação Básica com um vetor

```
void libera (Pilha* p) {
        free(p);
}
```

Exercício

- Desenvolva uma rotina para inverter a posição dos elementos de uma pilha P