Pilhas

Pilhas

- São listas onde a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo.

- Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.

- **Propriedade:** o último item inserido é o primeiro item que pode ser retirado da lista. São chamadas listas **lifo** (“last-in, first-out”).

TAD Pilhas: Operações

- Criar uma pilha P vazia
- Testar se P está vazia
- Obter o elemento do topo da pilha (sem eliminar)
- Inserir um novo elemento no topo de P (empilhar/push)
- Remover o elemento do topo de P (desempilhar/pop)

Pilhas: Implementações

- Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas.

- As duas representações mais utilizadas são as implementações por meio de **arranjos (estáticas)** e de **apontadores (dinâmicas)**.

Implementação de Pilhas: Arranjos

- Os itens da pilha são armazenados em posições contíguas de memória.

- Como as inserções e as retiradas ocorrem no topo da pilha, um ponteiro chamado **topo** é utilizado para controlar a posição do item no topo da pilha.


```
#define MAX 10

typedef struct {
   int valor;
} ITEM;

typedef struct {
    ITEM itens[MAX];
          int topo;
   } tpilha
```

Operações sobre Pilhas: Arranjos

- Criar uma pilha vazia

```
void criar (tpilha *p)
  {
    p->topo = -1;
}
```

- Verificar se a pilha está vazia

```
int vazia (tpilha *p){
   return (p->topo == -1);
}
```

- Verificar se a pilha está cheia

```
int cheia (tpilha *p){
   return (p->topo == MAX-1);
}
```

Operações sobre Pilhas: Arranjos

- Contar o número de elementos

```
int contar (tpilha *p)
  {
    return(p->topo+1);
}
```

- Obter o elemento do topo da pilha (sem eliminar)

```
int elemtopo (tpilha *p, ITEM *item){
   if (!cheia(p)) {
     *item = p->itens[p->topo];
      return 1;
     }
    return 0;
}
```

Operações sobre Pilhas: Arranjos

- Inserir um novo elemento no topo da pilha (empilhar)


```
int push (tpilha *p, ITEM *item)  {
   if (!cheia(pilha)){
      /* pilha não esta cheia*/
      p->topo++; /*incrementa o ponteiro*/
      p->itens[p->topo] = *item;
      return 1; }
   return 0;
}
```

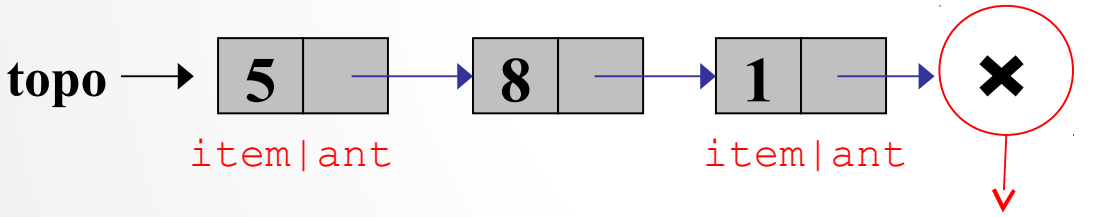
Operações sobre Pilhas: Arranjos

- Remover o elemento do topo da pilha (desempilhar)


```
int pop (tpilha *p, ITEM *item){
   if (!vazia(p)) {
        *item = p->itens[p->topo];
         p->topo--;
         return 1; }
   return 0;
}
```

Implementação de Pilhas: Apontadores

- Permite utilizar posições não contíguas de memória.
- Cada nó contém um item da pilha e um apontador para o nó seguinte.
- Toda pilha possui um apontador chamado **topo** que aponta para o primeiro nó da pilha.



Indicador do fim da pilha (referência **null**)

```
typedef struct {
    int valor;
 } ITEM;

typedef struct no{
    ITEM item;
    struct no *anterior;
 } tno;

typedef struct {
   int contador
   tno *topo;
} pilha
```

Operações sobre Pilhas: Apontadores

- Criar uma pilha vazia

```
void criar (pilha *p){
    p->contador = 0;
    p->topo = NULL;
 }
```

- Verifica se está vazia

```
int vazia (pilha *p) {
   return (p->topo==NULL);
 }
```

- Retorna o número de elementos

```
int contar (pilha *p) {
   return (p->contador);
 }
```

Operações sobre Pilhas: Apontadores

- Limpa a pilha

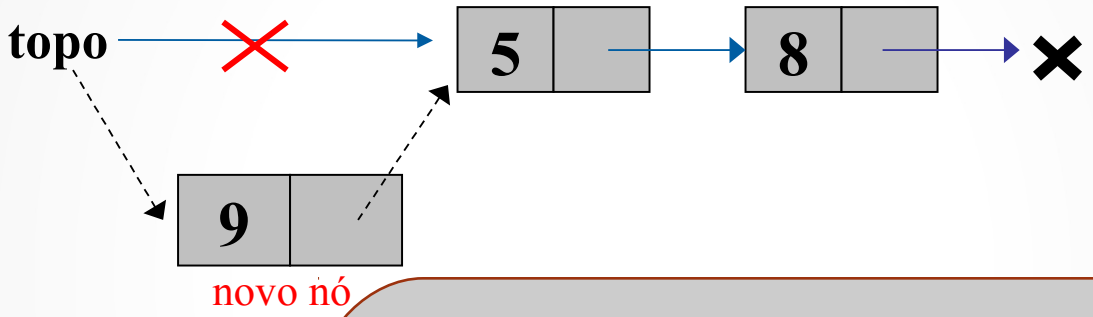
```
void limpar (pilha *p){
      tno *paux = p->topo;
      while (paux !=NULL)
          tno *prem = paux;
          paux = paux->anterior;
          free(prem);
      }
}
```

- Obter o elemento do topo da pilha (sem eliminar)

```
int elemtopo (pilha *p, ITEM *item) {
   if (!vazia(p))
       *item = p->topo->item;
        return 1;
    }
    return 0;
}
```

Operações sobre Pilhas: Apontadores

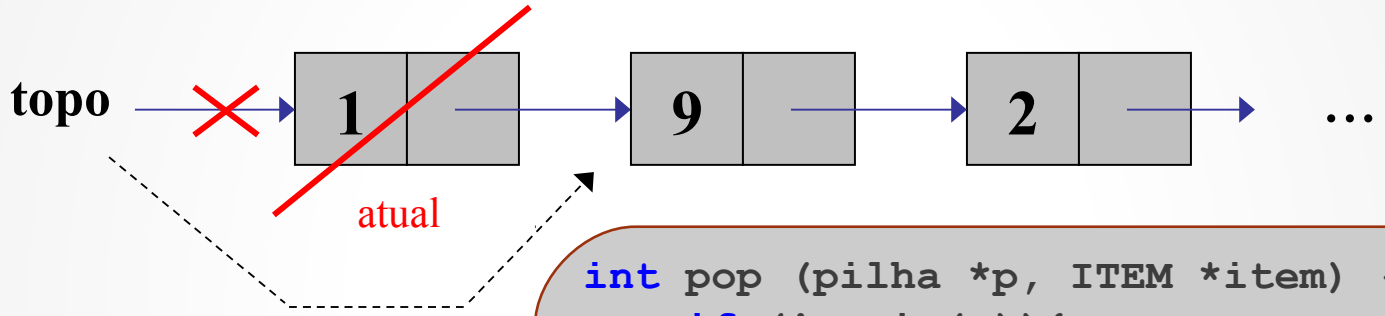
- Inserir um novo elemento no topo da pilha (empilhar)




```
int push (pilha *p, ITEM *item ){
    tno *pnovo = (tno *)malloc(sizeof(tno));
    if (pnovo!=NULL){
        pnovo->item = *item;
        pnovo->anterior = p->topo;
        p->topo = pnovo;
        p->contador++;
       return 1;
      }
      return 0;
 }
```

Operações sobre Pilhas: Apontadores

- Remover o elemento do topo da pilha (desempilhar),




```
int pop (pilha *p, ITEM *item) {
    if (!vazia(p)){
        *item = p->topo->item;
         tno *paux = p->topo;
         p->topo = p->topo->anterior;
         p->contador --;
         free(paux);
         return 1;
     }
   return 0;
}
```

Pilha Estática x Dinâmica

Operação	Estática	Dinâmica
cria	O(1)	O(1)
push	O(1)	O(1)
pop	O(1)	O(1)
topo	O(1)	O(1)
vazia	O(1)	O(1)
conta	O(1)	O(1) (contador)

Pilha Estática x Dinâmica

- Estática
  - Implementação simples
  - Tamanho da pilha definido a priori
- Dinâmica
  - Alocação dinâmica permite gerencia melhores estruturas cujo tamanho não é conhecido a prior ou que variam muito de tamanho

Pilha Estática x Dinâmica

- No caso geral de **listas** ordenadas, a maior vantagem da alocação dinâmica sobre a estática - se a memória não for problema - é a eliminação de deslocamentos na inserção ou eliminação dos elementos.

- No caso das **pilhas**, essas operações de deslocamento não ocorrem. Portanto, podemos dizer que a alocação estática é mais vantajosa na maioria das vezes. No entanto, a alocação dinâmica permite melhor gerenciamento de estruturas que podem aumentar ou diminuir de tamanho