Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/2

Estrutura de Dados

Estrutura de Dados
Pilha

Sumário

Definição

Pilha

> *Uma pilha (stack) é uma versão limitada de uma lista. Os novos elementos só podem ser adicionados no topo da pilha e também só podem ser removidos os nós do topo de uma pilha. Por esse motivo, pilha é conhecida como uma estrutura de dados último a entrar, primeiro a sair (Last-In,First-Out, ou LIFO)*
>
> (Deitel; Deitel, 2011)

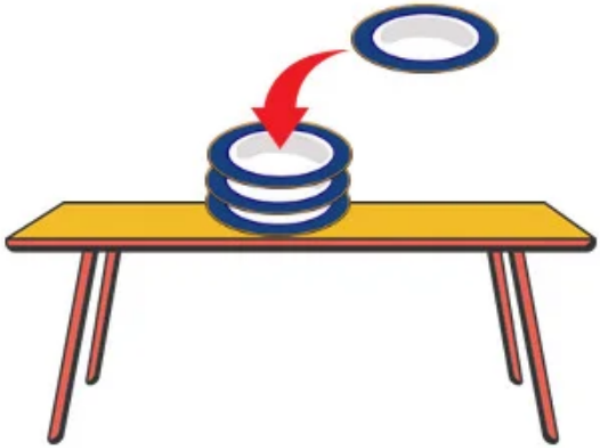
Atenção

Essa é a única forma de armazenar e recuperar em uma pilha; não é permitido acesso randômico a nenhum item específico.

Pilha físicas são utilizadas no dia-a-dia.
Por exemplo, uma pilha, caixas, livros ou quaisquer objetos.
Pilhas lógicas são utilizadas para resolver problemas computacionais como:

- **Chamada de Funções:** Cada chamada de função empilha o contexto de execução. Ao retornar, esse contexto é desempilhado.
- **Recursão:** A pilha é usada implicitamente para armazenar as chamadas recursivas pendentes.
- **Desfazer/Refazer em Editores:** Como em editores de texto, onde ações são empilhadas para desfazer ou refazer operações.
- **Avaliação de Expressões:** Especialmente em expressões na forma pós-fixa (notação polonesa reversa).

- **Navegação em Navegadores:** O histórico de páginas visitadas funciona como uma pilha (voltar/avançar).
- **Compiladores:** Analisadores sintáticos usam pilhas para checar correspondência de parênteses, blocos de código, etc.
- **Percurso em Árvores:** O percurso em profundidade utiliza pilhas para navegar pela estrutura recursivamente ou iterativamente.
- **Inversão de Listas:** Pilhas podem ser usadas para inverter a ordem dos elementos de uma lista de forma simples.



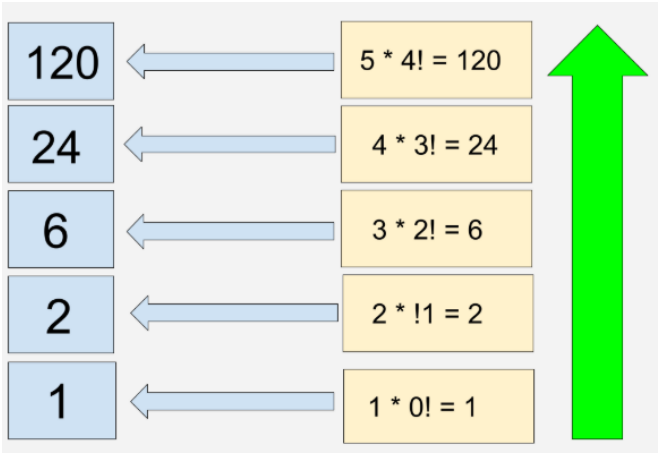


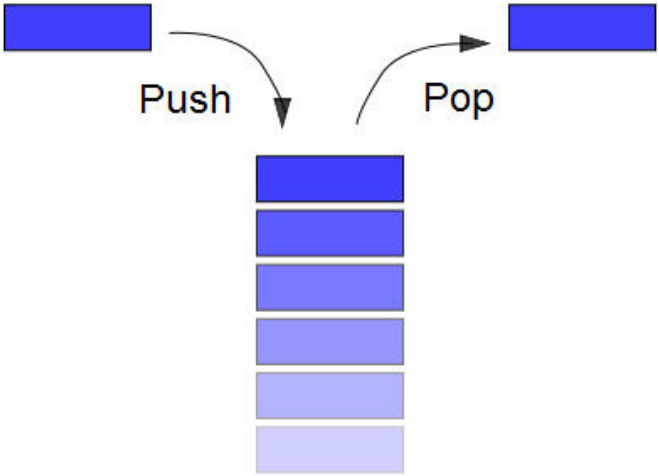
Figura 1: Exemplo de chamadas de funções recursivas para cálculo de fatorial.

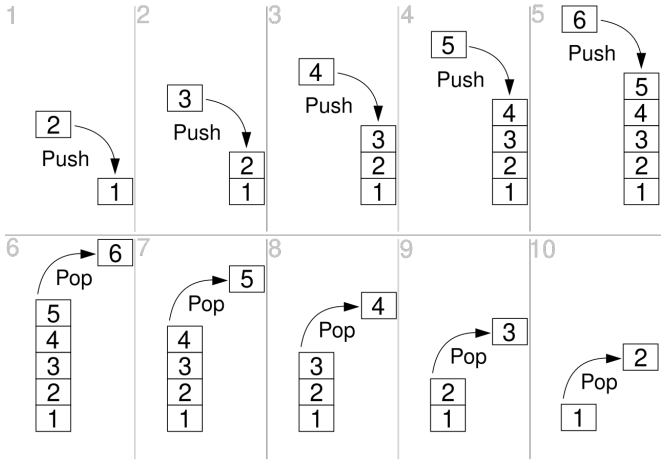
Operações

O funcionamento de uma pilha se resume em duas funções ou operações:

- As principais funções utilizadas para manipular uma pilha são `push` e `pop`.
- A função `push` cria um novo nó e o insere no topo da pilha.
- A função `pop` remove o nó do topo da pilha, libera a memória alocada e retorna o valor removido.

Em uma operação que remove um item da pilha, o item será destruído, se for necessário, o item deve ser armazenado.


Push
Pop


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1

Ação	Conteúdo da pilha
push(A)	A
push(B)	A B
push(C)	A B C
pop() devolve C	A B
push(D)	A B D
pop() devolve D	A B
pop() devolve B	A

Tabela 1: Operações em uma pilha e seus estados

- `inicializa_pilha(p)` — Cria uma pilha vazia.
- `pilha_vazia(p)` — Verifica se a pilha `p` está vazia.
- `pilha_cheia(p)` — Verifica se a pilha `p` está cheia.
- `visualizar_topo(p)` — Retorna o valor no topo da pilha `p`, sem removê-lo (somente leitura).
- `mostrar_pilha(p)` — Mostra na tela os elementos contidos na pilha `p` (função auxiliar).
- `desalocar_pilha(p)` — Libera o espaço de memória ocupado pela pilha `p`.

Pilha Simples

```c
typedef struct{
    int topo;
    int dados[CAPACIDADE_MAXIMA];
}Pilha;
```

Código 1: Estrutura da pilha.

```c
#define CAPACIDADE_MAXIMA 10

Pilha* inicializar();
int push(int dado, Pilha *pilha);
int pop(Pilha *pilha);
int cheia(Pilha *pilha);
int vazia(Pilha *pilha);
int mostrar(Pilha *pilha);
int visualizar_topo(Pilha *pilha);
```

Código 2: Exemplo de protótipos das funções.

```c
Pilha* inicializar(){
    Pilha *pilha = malloc(sizeof(Pilha));
    pilha->topo = -1;
    for (int i = 0; i < CAPACIDADE_MAXIMA; i++)
        pilha->dados[i] = -1;
    return pilha;
}
```

Código 3: Exemplo de inicialização.

```c
int push(int dado, Pilha *pilha){
    if(cheia(pilha))
        return 0;
    pilha->topo++;
    pilha->dados[pilha->topo] = dado;
    return 1;
}
```

Código 4: Exemplo de inserção.

```c
int pop(Pilha *pilha){
    if(vazia(pilha))
        return 0;
    int dado = pilha->dados[pilha->topo];
    pilha->topo--;
    return dado;
}
```

Código 5: Exemplo de remoção.

```c
int cheia(Pilha *pilha){
    return pilha->topo == CAPACIDADE_MAXIMA - 1;
}
```

Código 6: Exemplo de verificação de pilha cheia.

```c
int vazia(Pilha *pilha){
    return pilha->topo == -1;
}
```

Código 7: Exemplo de verificação de pilha vazia.

```c
int visualizar_topo(Pilha *pilha){
    if(vazia(pilha))
        return 0;
    printf(" [ %d ]", pilha->dados[pilha->topo]);
    return 1;
}
```

Código 8: Exemplo de visualozação do topo da pilha.

```c
#include "pilha.h"
int main(void)
{
    Pilha *pilha = inicializar();
    push(20, pilha);
    mostrar(pilha);
    push(10, pilha);
    mostrar(pilha);
    push(50, pilha);
    mostrar(pilha);
    pop(pilha);
    mostrar(pilha);
    push(5, pilha);
    mostrar(pilha);
    return 0;
}
```

Código 9: Exemplo de utilização da pilha.

- Por não fazer alocação dinâmica, esta versão da pilha não necessita de uma função para desalocação.
- Não é possível modificar a capacidade da pilha em tempo de execução.
- É necessário saber antecipadamente o número máximo de valores que precisam estar armazenados simultaneamente na pilha.

Pilha Encadeada

- **Pilha simples com `reallloc`**
  - Requer alocação de blocos contíguos de memória, o que pode ser difícil com o tempo devido à fragmentação.
  - Inserções e remoções podem exigir realocação ou deslocamento de elementos para manter a sequência.
  - Capacidade fixa ou necessidade de redimensionamento complexo (*realloc*), que pode ser custoso em tempo.
  - Pode desperdiçar espaço se a fila não estiver cheia (espaço pré-alocado não usado).
- **Pilha Encadeada**
  - Usa nós dinamicamente alocados ligados por ponteiros, aproveitando melhor a memória disponível.
  - Inserções e remoções são mais eficientes, pois não exigem deslocamento de elementos.
  - Não necessita de bloco contíguo de memória.

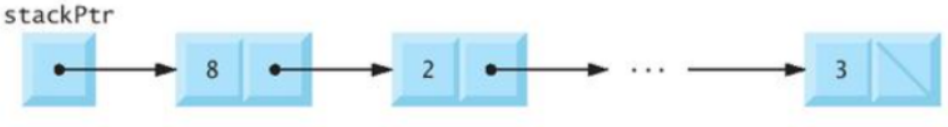


Figura 2: Operação enqueue (Deitel; Deitel, 2011).

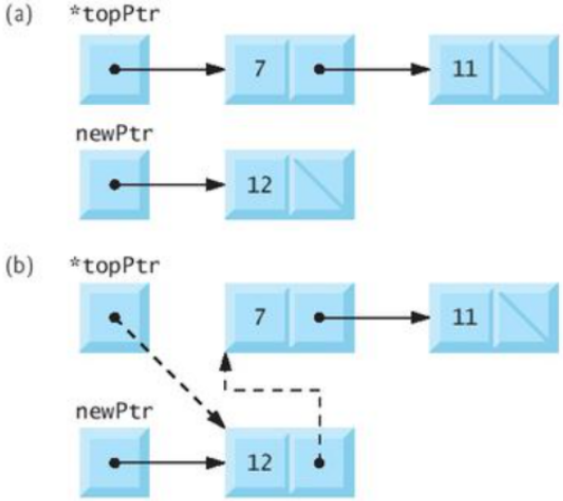

(a)
*topPtr
7
11
newPtr
12
(b)
*topPtr
7
11
newPtr
12

Figura 3: Operação push (Deitel; Deitel, 2011).

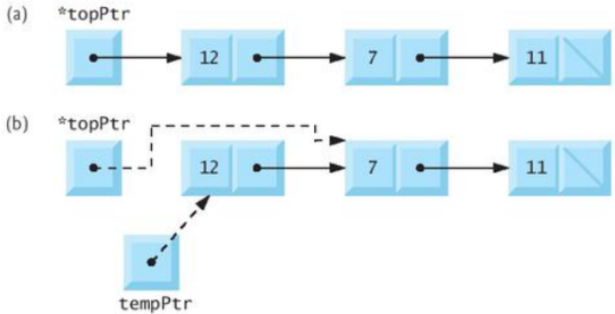


Figura 4: Operação pop (Deitel; Deitel, 2011).

(Deitel; Deitel, 2011) - Capítulo 12

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

SCHILDT, Herbert. **C Completo e Total: O Guia Definitivo para Programação em C**. 3. ed. São Paulo: Makron Books, 1996. ISBN 978-8534606928.

Estes slides estão protegidos por uma licença Creative Commons

Este modelo foi adaptado de Maxime Chupin.

Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/2

Estrutura de Dados

Estrutura de Dados
Pilha