

Pilhas

(IED-001)

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo



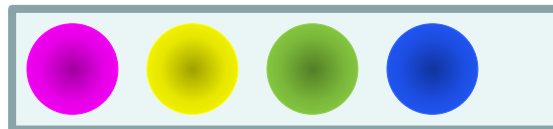
Pilhas

Pilha

é uma lista em que todas as operações de inserção, remoção e acesso são feitas num mesmo extremo, denominado **topo**.

Funcionamento:

- Quando um item é inserido numa pilha, ele é colocado em seu topo.
- Apenas o item no topo da pilha pode ser acessado ou removido.
- Essa política de acesso é denominada **LIFO** (*Last-In/First-Out*).



Exemplos de aplicação:

- Controle de acesso às páginas visitadas num navegador *web*.
- Controle de chamadas e retornos das funções num programa.



Pilha é útil em qualquer situação em que precisamos **inverter** a ordem de uma sequência!



Pilhas

Operações em pilhas

- **pilha(m)**
Cria e devolve uma pilha vazia **P**, com tamanho **m**.
- **vaziap(P)**
Devolve verdade se, e só se, a pilha **P** estiver vazia.
- **cheiap(P)**
Devolve verdade se, e só se, a pilha **P** estiver cheia.
- **empilha(x, P)**
Insere o item **x** no topo da pilha **P**.
- **desempilha(P)**
Remove e devolve o item que estiver no topo de **P**.
- **topo(P)**
Acessa e devolve o item que estiver no topo de **P**.
- **destroip(&P)**
Destrói a pilha **P**.

Exemplo 1. Operação, efeito e resultado

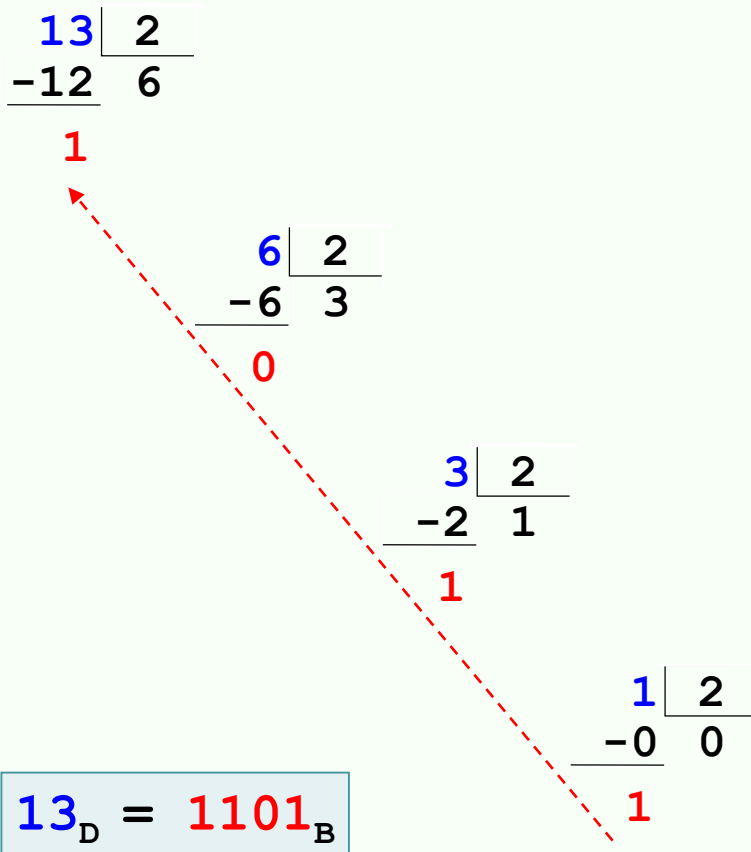
P = pilha(3)	[]	–
vaziap(P)	[]	1
cheiap(P)	[]	0
empilha(1, P)	[1]	–
empilha(2, P)	[1, 2]	–
empilha(3, P)	[1, 2, 3]	–
vaziap(P)	[1, 2, 3]	0
cheiap(P)	[1, 2, 3]	1
desempilha(P)	[1, 2]	3
desempilha(P)	[1]	2
empilha(desempilha(P), P)	[1]	–
empilha(topo(P), P)	[1, 1]	–
vaziap(P)	[1, 1]	0
cheiap(P)	[1, 1]	0
destroip(&P)	inexistente	–

Vamos começar supondo que esse tipo de dados está disponível no arquivo **pilha.h**!



Pilhas

Exemplo 2. Conversão em binário



```
#include <stdio.h>
#include "pilha.h"

int main(void) {
    int n;
    Pilha P = pilha(8*sizeof(int));
    printf("Decimal? ");
    scanf("%d",&n);
    do {
        empilha(n%2,P);
        n /= 2;
    } while( n>0 );
    printf("Binario: ");
    while( !vaziap(P) )
        printf("%d",desempilha(P));
    destroip(&P);
    return 0;
}
```

Usamos `<*.h>` para arquivos padrão e `"*.h"` para arquivos definidos pelo programador!



Pilhas

Exemplo 3. Inversão de cadeia

c:

R	O	M	A	\0
---	---	---	---	----

0 1 2 3 4
i

--	--	--	--	--

P:



A	M	O	R
---	---	---	---

```
#include <stdio.h>
#include "pilha.h"

int main(void) {
    char c[513];
    Pilha P = pilha(513);
    printf("Cadeia? ");
    gets(c);
    for(int i=0; c[i]; i++)
        empilha(c[i], P);
    printf("Inverso: ");
    while( !vaziap(P) )
        printf("%c", desempilha(P));
    puts("\n");
    destroip(&P);
    return 0;
}
```

Como o tipo **Pilha** e suas operações são definidos no arquivo **pilha.h**?

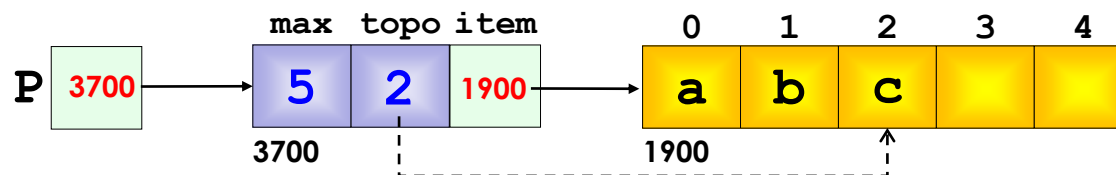


Pilhas

Exemplo 4. O tipo Pilha

```
typedef char Itemp;  
typedef struct pilha {  
    int    max;  
    int    topo;  
    Itemp *item;  
} *Pilha;
```

Pilha \equiv struct pilha *



Note que o tipo **Itemp** pode ser redefinido, em função da aplicação que usa o tipo **Pilha**!

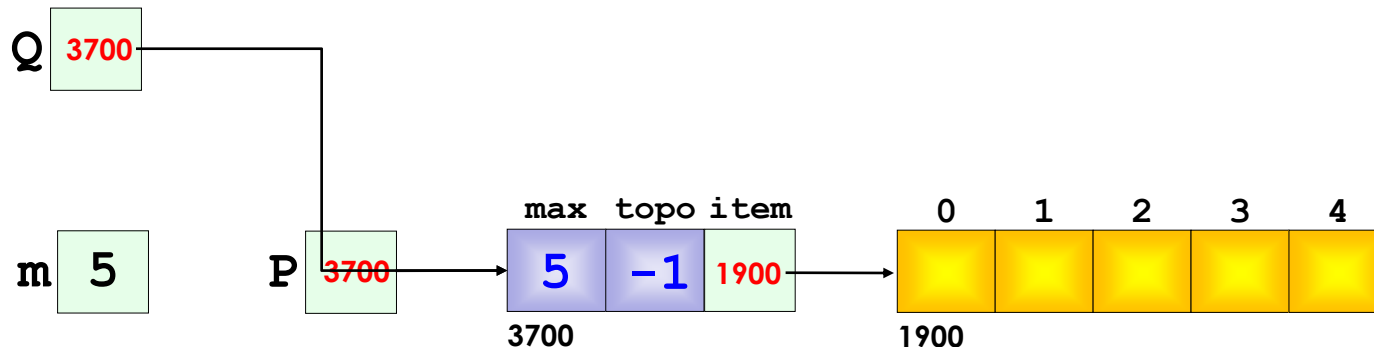


Pilhas

Exemplo 5. Criação de pilha vazia

```
➡ Pilha pilha(int m) {  
➡     Pilha P = malloc(sizeof(struct pilha));  
➡     P->max   = m;  
➡     P->topo  = -1;  
➡     P->item  = malloc(m*sizeof(Itemp));  
➡     return P;  
}
```

➡ Pilha Q = pilha(5);



Note que o nome **Pilha** identifica o tipo, enquanto o nome **pilha** identifica a função!

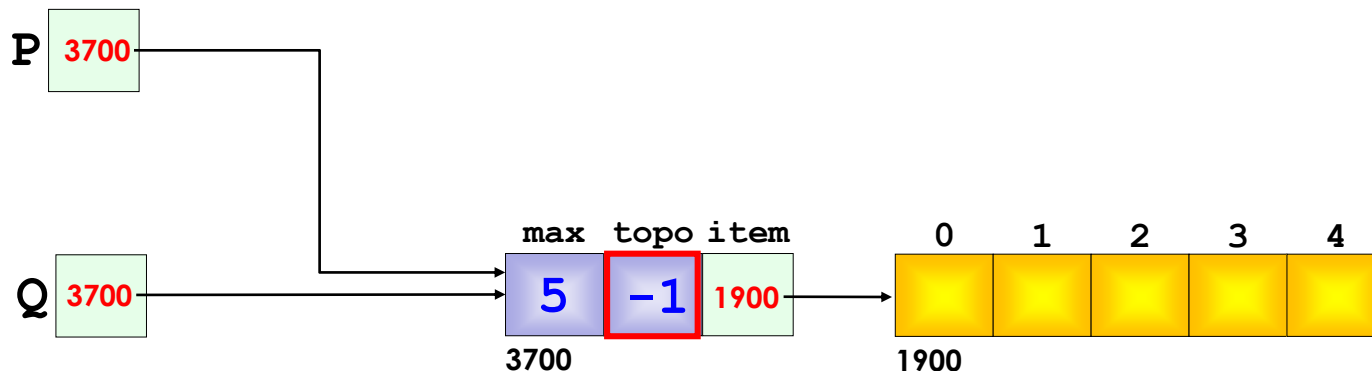


Pilhas

Exemplo 6. Teste de pilha vazia

```
➡ int vaziap(Pilha P) {  
➡     if( P->topo == -1 )  
➡         return 1;  
    else  
        return 0;  
}
```

➡ `vaziap(Q)`



A pilha está vazia quando o seu campo **topo** tem valor **-1**!

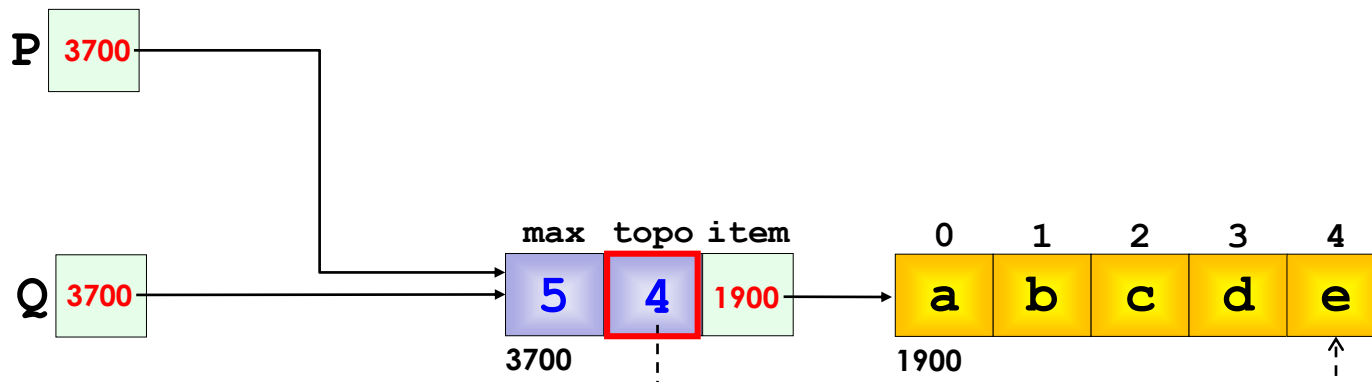


Pilhas

Exemplo 7. Teste de pilha cheia

```
➡ int cheia(Pilha P) {  
➡     if( P->topo == P->max-1 )  
➡         return 1;  
    else  
        return 0;  
}
```

➡ `cheia(Q)`



A pilha está cheia quando o seu campo **topo** tem valor **max-1**!

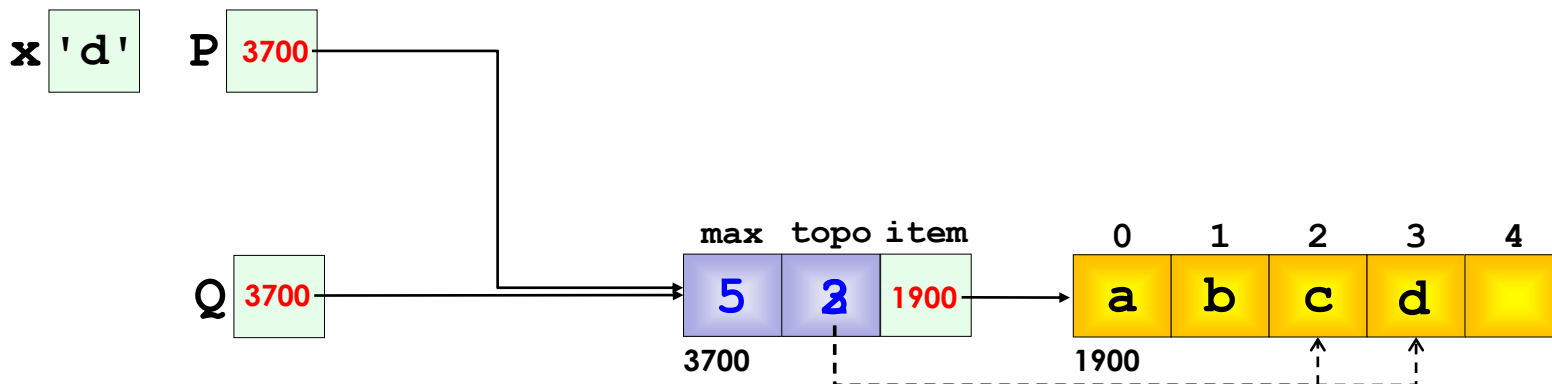


Pilhas

Exemplo 8. Inserção em pilha

```
void empilha(Item x, Pilha P) {  
    if( cheia(P) ) {  
        puts("pilha cheia!");  
        abort();  
    }  
    P->topo++;  
    P->item[P->topo] = x;  
}
```

→ `empilha('d', Q)`



A tentativa de **inserção** em pilha cheia causa o erro de **stack overflow!**

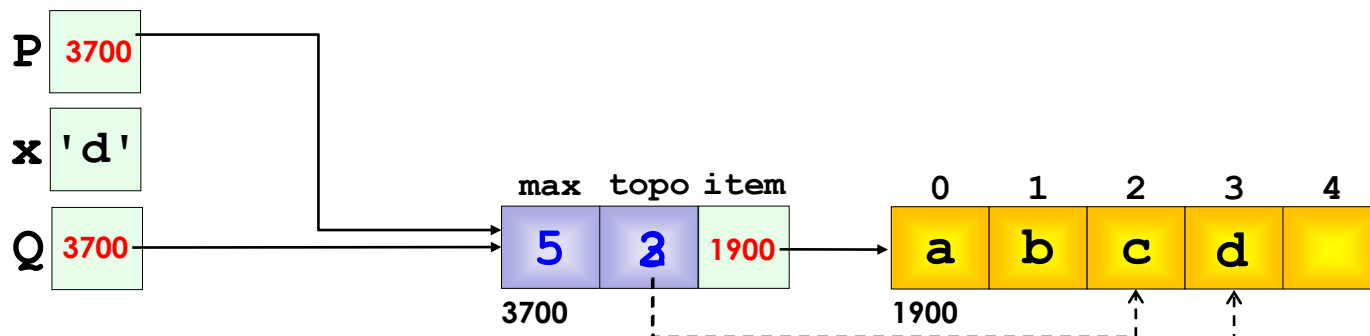


Pilhas

Exemplo 9. Remoção em pilha

```
➡ Itemp desempilha(Pilha P) {  
➡   if( vaziap(P) ) {  
       puts("pilha vazia!");  
       abort();  
   }  
➡   Itemp x = P->item[P->topo];  
➡   P->topo--;  
➡   return x;  
}
```

➡ **desempilha(Q)**



A tentativa de **remoção** em pilha vazia causa o erro de **stack underflow**!

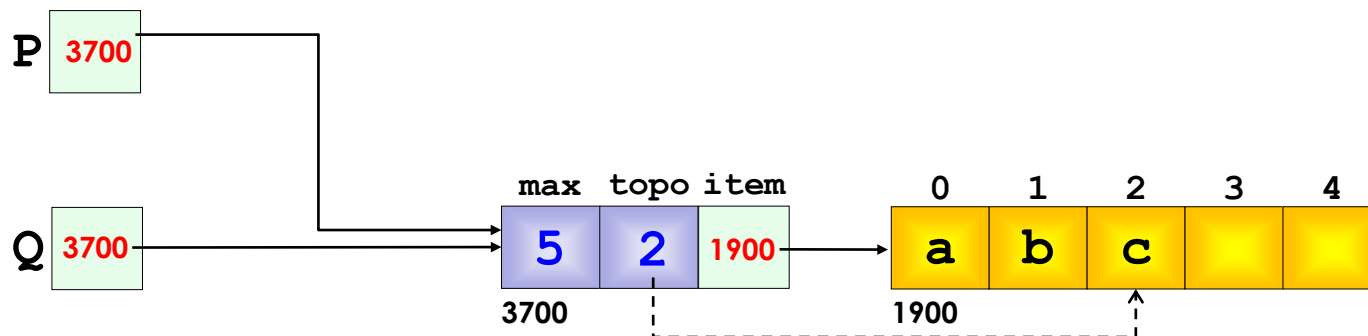


Pilhas

Exemplo 10. Acesso em pilha

```
➡ Itemp topo (Pilha P) {  
➡   if( vaziap(P) ) {  
       puts("pilha vazia!");  
       abort();  
   }  
➡   return P->item[P->topo];  
}
```

➡ topo(Q)



A tentativa de **acesso** em pilha vazia causa o erro de **stack underflow**!

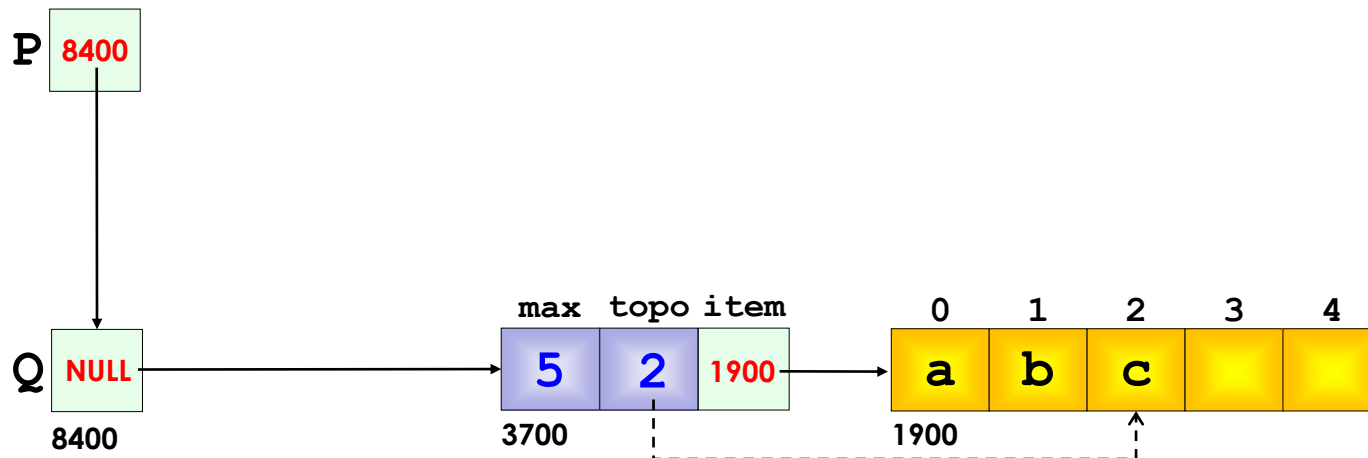


Pilhas

Exemplo 11. Destruição de pilha

```
void destroip(Pilha *P) {  
    free((*P)->item);  
    free(*P);  
    *P = NULL;  
}
```

→ `destroip(&Q)`



Após a destruição da pilha, ela não pode mais ser acessada!



Pilhas

Exercício 1. Ordenação crescente

Crie um programa que usa duas pilhas **A** e **B** para ordenar uma sequência de n números dados pelo usuário. A ideia é organizar a pilha **A** de modo que nenhum item seja empilhado sobre outro menor (use a pilha **B** apenas para manobra) e, depois, descarregar e exibir os itens da pilha **A**.

Exercício 2. Ordenação decrescente e sem repetição

Faça a alteração mínima necessária para que o programa do exercício anterior ordene os números em ordem decrescente, eliminando números repetidos.

Exercício 3. Inversão de palavras

Usando uma pilha, crie um programa para inverter a ordem das letras nas palavras de uma frase, sem inverter a ordem das palavras na frase. Por exemplo, se for digitada a frase "**apenas um teste**", o programa deverá produzir a seguinte saída: **sanepa mu etset**.

Exercício 4. Balanceamento de parênteses

Usando pilha, crie uma função para verificar se uma expressão composta apenas por chaves, colchetes e parênteses, representada por uma cadeia de caracteres, está ou não balanceada. Por exemplo, as expressões "[{ () } {}]" e "{ [([{ }])] }" estão balanceadas, mas as expressões "{ [()] }" e "{ [] () (] }" não estão.



Pilhas

Exercício 5. Pilha de strings

[versão 1]

Qual será a saída, se o usuário digite as cadeias "um", "dois" e "tres"? Por quê?

```
#include <stdio.h>
#include "pilha.h" // pilha de char *
int main(void) {
    Pilha P = pilha(5);
    char s[11];
    for(int i=1; i<=3; i++) {
        printf("? ");
        gets(s);
        empilha(s,P);
    }
    while( !vaziap(P) ) puts(desempilha(P));
    return 0;
}
```

Exercício 6. Pilha de strings

[versão 2]

Use `_strdup(s)`, declarada em `string.h`, para corrigir o programa do exercício anterior. Essa função duplica a cadeia `s` num área de memória, alocada pela função `malloc()`, e devolve o endereço dessa área. Depois de usada, essa cópia pode ser destruída com a função `free()`.

Fim

