Estrutura de Dados

Prof. Adriano Teixeira de Souza

Pilhas

- É uma das estruturas de dados mais simples
- A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo.
- Assim, quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo.

Pilhas
:: Aplicações

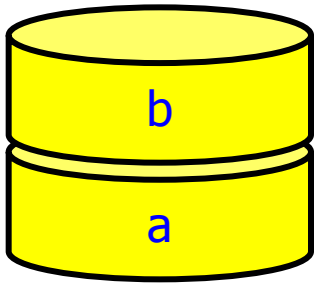
- Verificação de parênteses.
- Retirada de vagões de um trem.
- Retirada de mercadorias em um caminhão de entregas.

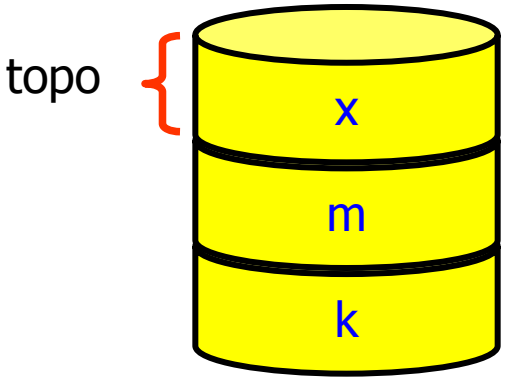
Pilhas

- 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).
- Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:
  - operação para empilhar (push) um novo elemento, inserindo-o no topo,
  - operação para desempilhar (pop) um elemento, removendo-o do topo

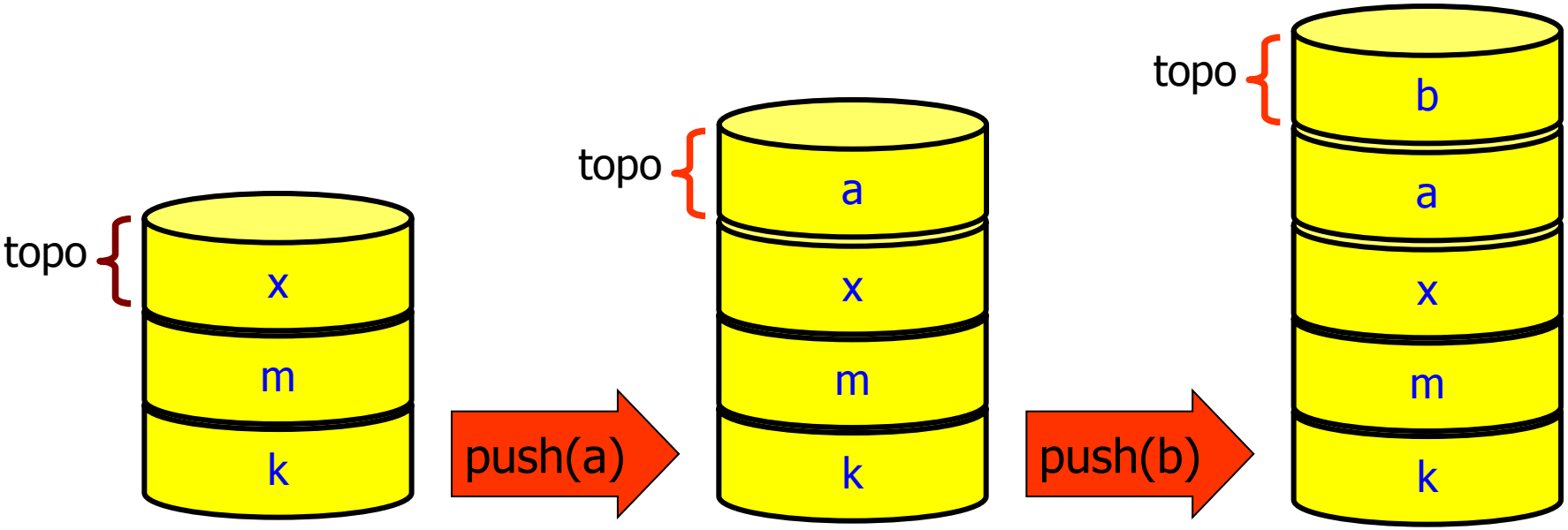
Pilhas
:: Push

push(b)



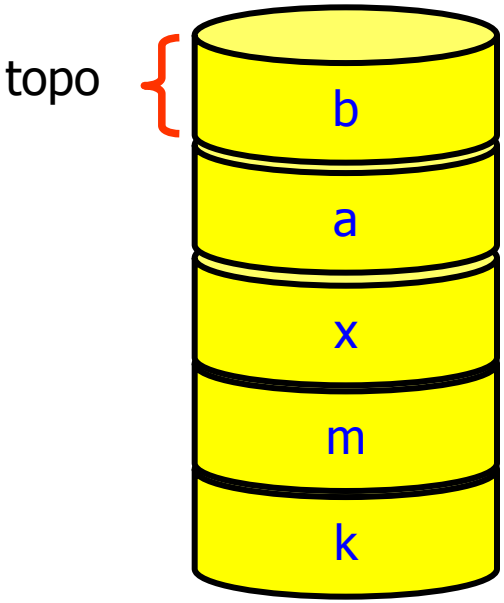


Pilhas
:: Push



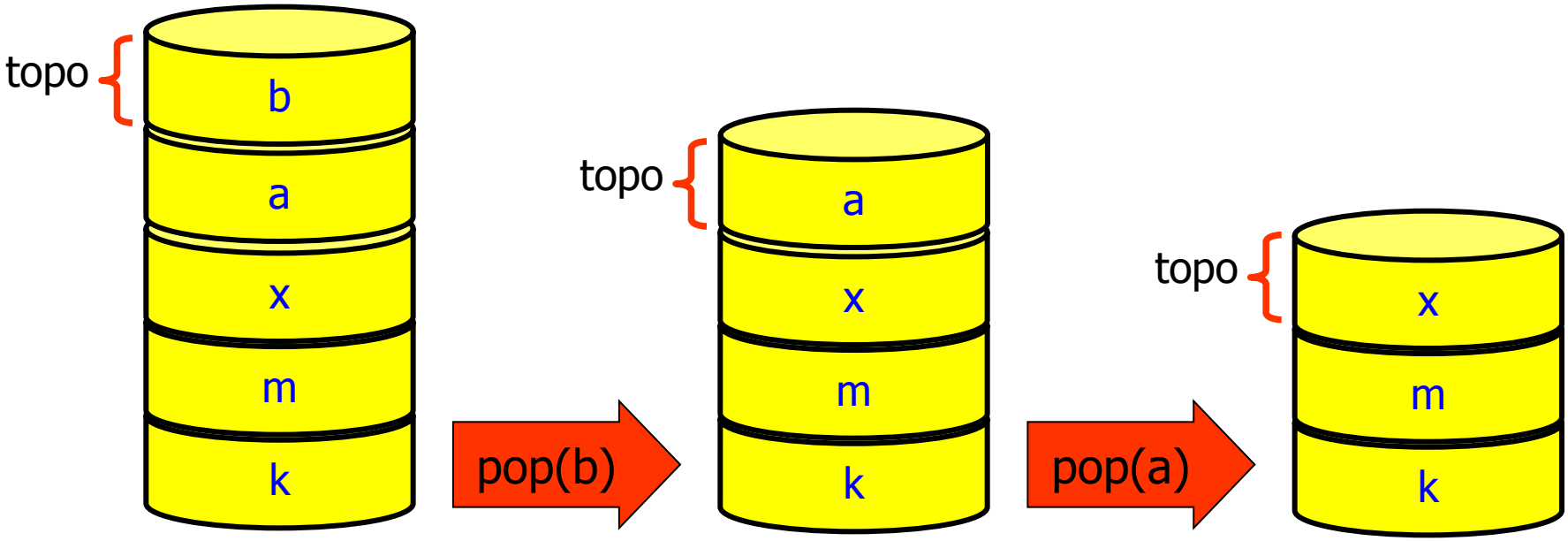
Pilhas
:: Pop

pop(b)



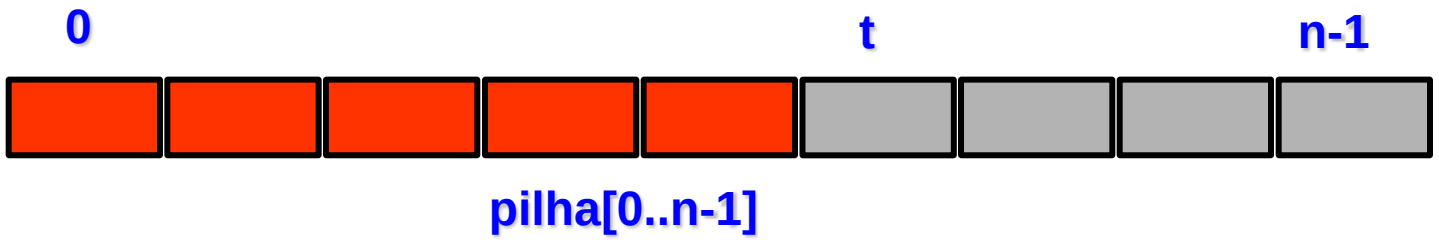
Pilhas

:: Pop



Pilhas
:: Implementação de pilha com vetor

- Supondo a pilha está armazenada em um vetor pilha[0..n-1].
- A parte do vetor ocupada pela pilha será:



Pilhas
:: Implementação de pilha com vetor

```
#define MAX 50

typedef struct {
   int n;
   float vet[MAX];
} Pilha;
```

Pilhas
:: Cria estrutura de pilha

```
Pilha* cria(void)
{
   Pilha* p;
   p = (Pilha*) malloc(sizeof(Pilha));
   p->n = 0;/*Inicializa com zero elementos*/
   return p;
}
```

Pilhas
:: Inserir o elemento do topo – push()

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

Pilhas
:: Remover o elemento do topo – pop()

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

Pilhas
:: Verificar se a pilha está vazia

```
int vazia(Pilha* p)
{
    return (p->n == 0);
}
```

Pilhas
:: Liberar a estrutura de pilha

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

Pilhas
:: Imprime estrutura de pilha

```
void imprime (Pilha* p) {
        int i;
        for (i=p->n-1; i>=0; i--)
                printf("%f\n",p->vet[i]);
}
```

Pilhas
:: Teste

```
main(){
        Pilha* p = cria();
        push(p,20.0);
        push(p,20.8);
        push(p,20.3);
        push(p,44.5);
        push(p,33.3);
        push(p,20.9);
        imprime (p);
        printf ("Retirado: %4.2f\n", pop(p));
        printf ("Retirado: %4.2f\n", pop(p));
        printf ("Configuracao da fila:\n");
        imprime (p);
        libera (p);
      system("pause");
}
```

Implementação de pilha com lista

- Quando o número máximo de elementos que serão armazenados na pilha não é conhecido, devemos implementar a pilha usando uma estrutura de dados dinâmica, no caso, empregando uma lista encadeada.
- Os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

Pilhas
:: Implementação de pilha com estruturas

```
typedef struct {
    float        info;
    struct No*   anterior;
} No;

typedef struct {
   No* topo;
} Pilha;
```

Pilhas
:: 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

Pilhas
:: Criar uma estrutura de pilha

```
Pilha* cria(void)
{
   Pilha *p;
   p = (Pilha*) malloc(sizeof(Pilha));
   p->topo = NULL;
   return p;
}
```

Pilhas
:: Inserir o elemento do topo – push()

```
Pilha* push(Pilha *p, float v)
{
   No* aux;
   aux = (No*) malloc(sizeof(No));
   aux->info = v;
   aux->anterior = p->topo;
   p->topo = aux;
   return aux;
}
```

Pilhas
:: Remover o elemento do topo – pop()

```
float pop(Pilha *p)
{
   float v;
   No* aux;
   if (vazia(p))
   {
      printf("Pilha vazia.");
      exit(1); /*aborta o programa*/
   }
   v      = p->topo->info;
   aux    = p->topo;
   p->topo = aux->anterior;
   free(aux);
    return v;
}
```

Pilhas
:: Verificar se a pilha está vazia

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

Pilhas
:: Liberar a estrutura de pilha

```
void libera(Pilha *p)
{
    No* q = p->topo;
    while (q != NULL)
    {
        No *t = q->anterior;
        free(q);
        q = t;
    }
    free(p);
}
```

Pilhas
:: Imprime estrutura de pilha

```
void imprime (Pilha* p) {
        No* q;
        for (q=p->topo; q!=NULL; q=q->anterior)
            printf("%4.2f\n",q->info);
}
```

Pilhas
:: Teste

```
main(){
        Pilha* p = cria();
        push(p,20.0);
        push(p,20.8);
        push(p,20.3);
        push(p,44.5);
        push(p,33.3);
        push(p,20.9);
        imprime (p);
        printf ("Retirado: %4.2f\n", pop(p));
        printf ("Retirado: %4.2f\n", pop(p));
        printf ("Configuracao da fila:\n");
        imprime (p);
        libera (p);
      system("pause");
}
```

Exercício

- Utilizando o algorítimo anteriormente apresentado implemente um programa que insira dados em uma pilha A e em seguida remova-os elementos da pilha A e insira-os na pilha B com sua ordem invertida.