

Pilhas

Aula 17

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

- 1 Introdução
- 2 Definição
- 3 Operações básicas em alocação sequencial
- 4 Operações básicas em alocação encadeada
- 5 Exercícios

- ▶ **lista linear especial**
- ▶ política de inserções e remoções bem definida
- ▶ inserção e remoção são as únicas operações
- ▶ inserção e remoção são realizadas em um único extremo

- ▶ lista linear especial
- ▶ política de inserções e remoções bem definida
- ▶ inserção e remoção são as únicas operações
- ▶ inserção e remoção são realizadas em um único extremo

- ▶ lista linear especial
- ▶ política de inserções e remoções bem definida
- ▶ inserção e remoção são as únicas operações
- ▶ inserção e remoção são realizadas em um único extremo

- ▶ lista linear especial
- ▶ política de inserções e remoções bem definida
- ▶ inserção e remoção são as únicas operações
- ▶ inserção e remoção são realizadas em um único extremo

- ▶ **pilha** é uma lista linear tal que as operações de inserção e remoção são realizadas em um único extremo dessa estrutura de dados
- ▶ funcionamento dessa estrutura pode ser comparado a qualquer pilha de objetos que usamos com frequência, por exemplo, uma pilha de pratos
- ▶ o extremo onde ocorrem as operações de inserção e remoção é chamado de **topo** da pilha
- ▶ inserção e remoção também são chamadas de empilhamento e desempilhamento
- ▶ nenhuma outra operação (p. ex. busca) é realizada, a não ser em casos específicos

- ▶ **pilha** é uma lista linear tal que as operações de inserção e remoção são realizadas em um único extremo dessa estrutura de dados
- ▶ funcionamento dessa estrutura pode ser comparado a qualquer pilha de objetos que usamos com frequência, por exemplo, uma pilha de pratos
- ▶ o extremo onde ocorrem as operações de inserção e remoção é chamado de **topo** da pilha
- ▶ inserção e remoção também são chamadas de empilhamento e desempilhamento
- ▶ nenhuma outra operação (p. ex. busca) é realizada, a não ser em casos específicos

- ▶ **pilha** é uma lista linear tal que as operações de inserção e remoção são realizadas em um único extremo dessa estrutura de dados
- ▶ funcionamento dessa estrutura pode ser comparado a qualquer pilha de objetos que usamos com frequência, por exemplo, uma pilha de pratos
- ▶ o extremo onde ocorrem as operações de inserção e remoção é chamado de **topo** da pilha
- ▶ inserção e remoção também são chamadas de empilhamento e desempilhamento
- ▶ nenhuma outra operação (p. ex. busca) é realizada, a não ser em casos específicos

- ▶ **pilha** é uma lista linear tal que as operações de inserção e remoção são realizadas em um único extremo dessa estrutura de dados
- ▶ funcionamento dessa estrutura pode ser comparado a qualquer pilha de objetos que usamos com frequência, por exemplo, uma pilha de pratos
- ▶ o extremo onde ocorrem as operações de inserção e remoção é chamado de **topo** da pilha
- ▶ inserção e remoção também são chamadas de empilhamento e desempilhamento
- ▶ nenhuma outra operação (p. ex. busca) é realizada, a não ser em casos específicos

- ▶ **pilha** é uma lista linear tal que as operações de inserção e remoção são realizadas em um único extremo dessa estrutura de dados
- ▶ funcionamento dessa estrutura pode ser comparado a qualquer pilha de objetos que usamos com frequência, por exemplo, uma pilha de pratos
- ▶ o extremo onde ocorrem as operações de inserção e remoção é chamado de **topo** da pilha
- ▶ inserção e remoção também são chamadas de empilhamento e desempilhamento
- ▶ nenhuma outra operação (p. ex. busca) é realizada, a não ser em casos específicos

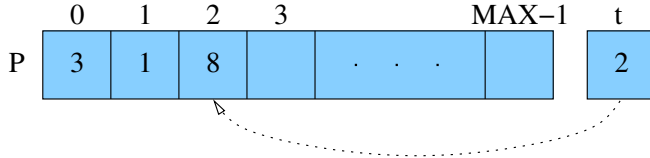
Operações básicas em alocação sequencial

- ▶ pilha armazenada em um segmento $P[0..t]$ de um vetor $P[0..MAX-1]$, com $-1 \leq t < MAX$
- ▶ t é o índice que define o topo da pilha e onde se encontra o último elemento empilhado

Operações básicas em alocação sequencial

- ▶ pilha armazenada em um segmento $P[0..t]$ de um vetor $P[0..MAX-1]$, com $-1 \leq t < MAX$
- ▶ t é o índice que define o topo da pilha e onde se encontra o último elemento empilhado

Operações básicas em alocação sequencial



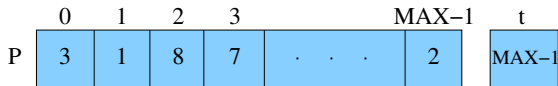
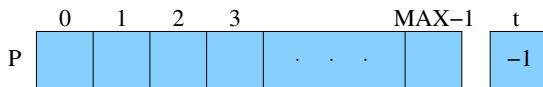
Operações básicas em alocação sequencial

- ▶ uma pilha está **vazia** se $t = -1$
- ▶ uma pilha está **cheia** se $t = \text{MAX}-1$

Operações básicas em alocação sequencial

- ▶ uma pilha está **vazia** se $t = -1$
- ▶ uma pilha está **cheia** se $t = \text{MAX}-1$

Operações básicas em alocação sequencial



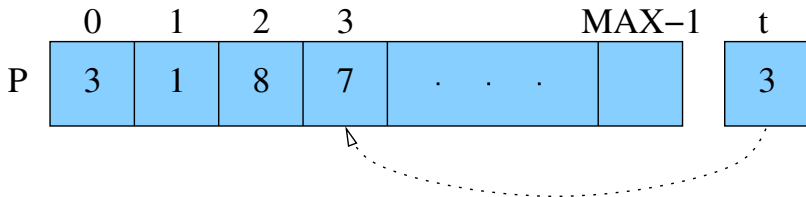
Operações básicas em alocação sequencial

- ▶ declaração e inicialização de uma pilha em alocação sequencial

```
int t, P[MAX];  
t = -1;
```

Operações básicas em alocação sequencial

- ▶ inserção de uma chave na pilha

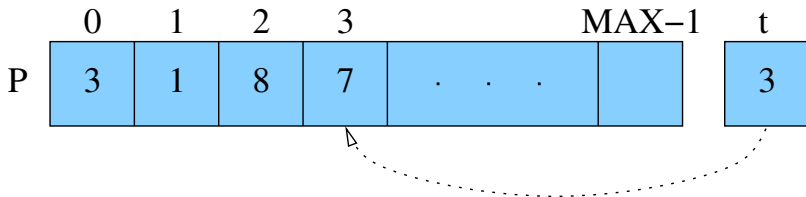


Operações básicas em alocação sequencial

```
void empilha_seq(int *t, int P[MAX], int y)
{
    if (*t != MAX - 1) {
        (*t)++;
        P[*t] = y;
    }
    else
        printf("Pilha cheia!\n");
}
```

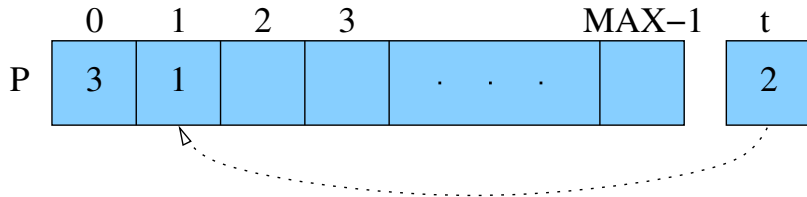
Operações básicas em alocação sequencial

- remoção de uma chave na pilha



Operações básicas em alocação sequencial

- remoção de uma chave na pilha



Operações básicas em alocação sequencial

```
int desempilha_seq(int *t, int P[MAX])
{
    int r;

    if (*t != -1) {
        r = P[*t];
        (*t)--;
    } else {
        r = INT_MIN;
        printf("Pilha vazia!\n");
    }
    return r;
}
```

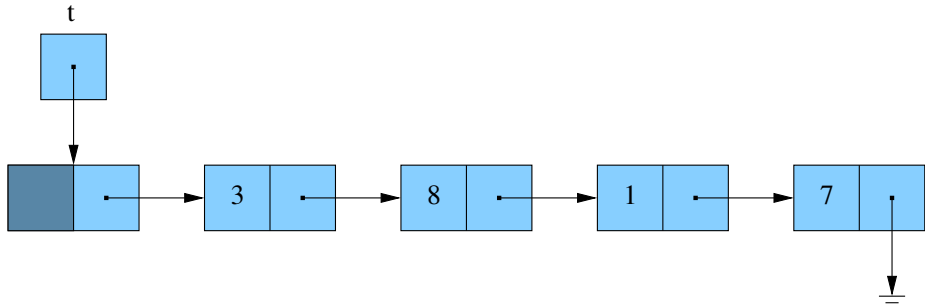
- ▶ conversão de notação de expressões aritméticas (notação *prefixa*, *infixa* e *posfixa*)
- ▶ implementação da pilha de execução de um programa no sistema operacional
- ▶ análise léxica de um programa realizada pelo compilador
- ▶ Exemplo: queremos saber se uma seqüência de caracteres tem a forma aZb (comprimento de a = comprimento de b)

- ▶ conversão de notação de expressões aritméticas (notação *prefixa*, *infixa* e *posfixa*)
- ▶ implementação da pilha de execução de um programa no sistema operacional
- ▶ análise léxica de um programa realizada pelo compilador
- ▶ Exemplo: queremos saber se uma seqüência de caracteres tem a forma aZb (comprimento de a = comprimento de b)

- ▶ conversão de notação de expressões aritméticas (notação *prefixa*, *infixa* e *posfixa*)
- ▶ implementação da pilha de execução de um programa no sistema operacional
- ▶ análise léxica de um programa realizada pelo compilador
- ▶ Exemplo: queremos saber se uma seqüência de caracteres tem a forma aZb (comprimento de a = comprimento de b)

- ▶ conversão de notação de expressões aritméticas (notação *prefixa*, *infixa* e *posfixa*)
- ▶ implementação da pilha de execução de um programa no sistema operacional
- ▶ análise léxica de um programa realizada pelo compilador
- ▶ Exemplo: queremos saber se uma seqüência de caracteres tem a forma aZb (comprimento de a = comprimento de b)

Operações básicas em alocação encadeada



Operações básicas em alocação encadeada

- ▶ tipo célula:

```
typedef struct cel {  
    int chave;  
    struct cel *prox;  
} celula;
```

- ▶ declaração e inicialização de uma pilha vazia em alocação encadeada com cabeça:

```
celula *t;  
t = (celula *) malloc(sizeof (celula));  
t->prox = NULL;
```

- ▶ declaração e inicialização de uma pilha sem cabeça:

```
celula *t;  
t = NULL;
```

Operações básicas em alocação encadeada

► tipo célula:

```
typedef struct cel {  
    int chave;  
    struct cel *prox;  
} celula;
```

► declaração e inicialização de uma pilha vazia em alocação encadeada com cabeça:

```
celula *t;  
t = (celula *) malloc(sizeof (celula));  
t->prox = NULL;
```

► declaração e inicialização de uma pilha sem cabeça:

```
celula *t;  
t = NULL;
```

Operações básicas em alocação encadeada

- ▶ tipo célula:

```
typedef struct cel {  
    int chave;  
    struct cel *prox;  
} celula;
```

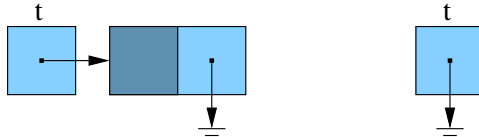
- ▶ declaração e inicialização de uma pilha vazia em alocação encadeada com cabeça:

```
celula *t;  
t = (celula *) malloc(sizeof (celula));  
t->prox = NULL;
```

- ▶ declaração e inicialização de uma pilha sem cabeça:

```
celula *t;  
t = NULL;
```

Operações básicas em alocação encadeada



Operações básicas em alocação encadeada

```
void empilha_enc_C(int y, celula *t)
{
    celula *nova;

    nova = (celula *) malloc(sizeof (celula));
    nova->chave = y;
    nova->prox = t->prox;
    t->prox = nova;
}
```

Operações básicas em alocação encadeada

```
int desempilha_enc_C(celula *t)
{
    int x;
    celula *p;

    if (t->prox != NULL) {
        p = t->prox;
        x = p->chave;
        t->prox = p->prox;
        free(p);
        return x;
    }
    else {
        printf("Pilha vazia!\n");
        return INT_MIN;
    }
}
```

1. Considere uma pilha em alocação encadeada sem cabeça. Escreva as funções para empilhar um elemento na pilha e desempilhar um elemento da pilha.
2. Uma palavra é um **palíndromo** se a seqüência de caracteres que a constitui é a mesma quer seja lida da esquerda para a direita ou da direita para a esquerda. Por exemplo, as palavras RADAR e MIRIM são palíndromos. Escreva um programa eficiente para reconhecer se uma dada palavra é palíndromo.

3. Um estacionamento possui um único corredor que permite dispor 10 carros. Existe somente uma única entrada/saída do estacionamento em um dos extremos do corredor. Se um cliente quer retirar um carro que não está próximo à saída, todos os carros impedindo sua passagem são retirados, o cliente retira seu carro e os outros carros são recolocados na mesma ordem que estavam originalmente. (*continua*)

3. (*continuação*) Escreva um algoritmo que processe o fluxo de chegada/saída deste estacionamento. Cada entrada para o algoritmo contém uma letra **E** para entrada ou **S** para saída, e o número da placa do carro. Considere que os carros chegam e saem pela ordem especificada na entrada. O algoritmo deve imprimir uma mensagem sempre que um carro chega ou sai. Quando um carro chega, a mensagem deve especificar se existe ou não vaga para o carro no estacionamento. Se não existe vaga, o carro não entra no estacionamento e vai embora. Quando um carro sai do estacionamento, a mensagem deve incluir o número de vezes que o carro foi movimentado para fora da garagem, para permitir que outros carros pudessem sair.

4. Considere o problema de decidir se uma dada seqüência de parênteses e chaves é bem-formada. Por exemplo, a seqüência abaixo:

(() { () })

é bem-formada, enquanto que a seqüência

({) }

é malformada.

Suponha que a seqüência de parênteses e chaves está armazenada em uma cadeia de caracteres **s**. Escreva uma função **bem_formada** que receba a cadeia de caracteres **s** e devolva **1** se **s** contém uma seqüência bem-formada de parênteses e chaves e devolva **0** se a seqüência está malformada.

5. Expressões aritméticas podem ser representadas usando diferentes notações. Costumeiramente, trabalhamos com expressões aritméticas em notação *infixa*, isto é, na notação em que os operadores binários aparecem entre os operandos. Outra notação freqüentemente usada em compiladores é a notação *posfixa*, aquela em que os operadores aparecem depois dos operandos. Essa notação é mais econômica por dispensar o uso de parênteses para alteração da prioridade das operações. Exemplos de expressões nas duas notações são apresentados abaixo.

notação <i>infixa</i>	notação <i>posfixa</i>
$(A + B * C)$	$A B C * +$
$(A * (B + C) / D + E)$	$A B C + * D / E -$
$(A + B * C / D * E - F)$	$A B C * D / E * + F -$

(*continua*)

5. (*continuação*) Escreva uma função que receba uma cadeia de caracteres contendo uma expressão aritmética em notação *infixa* e devolva uma cadeia de caracteres contendo a mesma expressão aritmética em notação *posfixa*. Considere que a cadeia de caracteres da expressão *infixa* contém apenas letras, parênteses e os símbolos $+$, $-$, $*$ e $/$. Considere também que cada variável tem apenas uma letra e que a cadeia de caracteres *infixa* sempre está envolvida por um par de parênteses.

6. Suponha que exista um único vetor M de células de um tipo pilha pré-definido, com um total de **MAX** posições. Este vetor fará o papel da memória do computador. Este vetor M será compartilhado por duas pilhas em alocação seqüencial. Implemente eficientemente as operações de empilhamento e desempilhamento para as duas pilhas de modo que nenhuma das pilhas estoure sua capacidade de armazenamento, a menos que o total de elementos em ambas as pilhas seja **MAX**.

