Pilha, Fila
Algoritmos e Estrutura de Dados
Pilha e Fila
(Alocação Dinâmica)

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Roteiro da Aula

Introdução

Estrutura Nodo

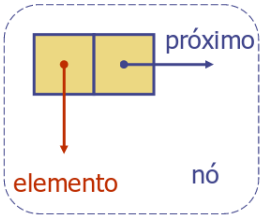
- A entidade elementar de uma estrutura de dados é o nodo (nó).
- Um nodo é diferenciado pelo seu endereço relativo dentro da estrutura e pode ser constituído de um ou vários campos.
- Cada campo de um nodo armazena uma informação, um dado, que pode ser do tipo numérico, alfanumérico, lógicos, imagens, sons, enfim, qualquer informação que possa ser manipulada.
- Todos os nodos de uma mesma estrutura devem ter a mesma configuração.

Introdução

Estrutura Nodo

Cada nodo armazena:

- Um elemento.
  - pode ser um inteiro, uma string, um vetor, um registro, não importa.
  - Aqui é denominado apenas **item**.
- Uma ligação para o próximo nodo.
  - Em termos de implementação, trata-se de um ponteiro do tipo nodo, o qual denominaremos **prox**.



Primeiramente, vamos implementar o tipo de dados `nodo`. Vamos analisar o pseudo-código apresentado nas aulas anteriores.

Introdução

Estrutura Nodo

Algoritmo 1: Nodo

```
1 início
2     registro {
3         Inteiro: item;
4         Ponteiro Nodo: prox;
5     } Nodo;
```

Introdução

Estrutura Nodo

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 typedef struct nodo_t {
2     int item;
3     struct nodo_t *prox;
4 } nodo;
```

Vamos chamar o arquivo que contém esse código de `nodo.h`.

Roteiro da Aula

Exercício 2

Estrutura Pilha

Neste exercício, vamos implementar a estrutura de dados **Pilha**:

- Nessa estrutura existem dois ponteiros:
  - um aponta para o **topo** da Pilha.
  - o outro que aponta para o **fundo** da Pilha.
- No topo da Pilha existe um nodo vazio (denominado nodo **cabeça**).
- O ponteiro topo sempre aponta para o nodo cabeça.
- Quando a Pilha está vazia, fundo e topo apontam para o nodo cabeça.

Exercício 2

Pilha

Considere o pseudo-código a seguir:

Algoritmo 2: Pilha

```
1 início
2     registro {
3         Ponteiro Nodo: topo, fundo;
4     } Pilha;
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 #include "nodo.h"
2 typedef struct {
3     nodo *topo, *fundo;
4 } pilha;
```

Vamos chamar o arquivo que contém esse código de `pilha.h`.

Exercício 2

Pilha

A seguir, vamos implementar cada uma das seguintes funções:

1. CriaPilhaVazia(S)
2. PilhaVazia(S)
3. Empilha(S,x)
4. Desempilha(S)
5. ApagaPilha(S)

Exercício 2

Pilha

Vamos adicionar o protótipo de cada uma das funções no arquivo `pilha.h`.

```
1 void criaPilhaVazia(pilha *s);
2 int pilhaVazia(pilha *s);
3 void empilha(pilha *s, int x);
4 int desempilha(pilha *s);
5 void apagaPilha(pilha *s);
```

Agora, vamos implementar cada uma dessas funções.

Exercício 2

Pilha

- Primeiramente, vamos criar um arquivo chamado `pilha.c` que deve conter a implementação dessas funções.
- Para evitar que o compilador não encontre determinada função, vamos incluir o protótipo das funções, inserindo a linha de código a seguir no início do arquivo `pilha.c`:

```
1 #include "pilha.h"
```

- A seguir, é apresentado a implementação das funções, que devem ser incluídas no arquivo `pilha.c`.

Exercício 2

Pilha

Algoritmo 3: CriaPilhaVazia

Entrada: Pilha S.

```
1 início
2     S.topo ← ALOCA_NODO()
3     S.fundo ← S.topo
4     S.topo.prox ← NULL
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void criaPilhaVazia(pilha *s) {
2     s->topo = malloc(sizeof(nodo));
3     s->fundo = s->topo;
4     s->topo->prox = NULL;
5 }
```

Exercício 2

Pilha

Algoritmo 4: PilhaVazia

Entrada: Pilha S.
Saída: Booleano (V ou F) indicando se S está vazia.

```
1 início
2     se (S.topo = S.fundo) então
3         retorna Verdadeiro
4     senão
5         retorna Falso
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 int pilhaVazia(pilha *s) {
2     return s->topo == s->fundo;
3 }
```

Exercício 2

Pilha

Algoritmo 5: Empilha

Entrada: Pilha S, item x.

```
1 início
2     novo ← ALOCA_NODO()
3     novo.prox ← S.topo
4     S.topo.item ← x
5     S.topo ← novo
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```c
void empilha(pilha *s, int x) {
    nodo *novo = malloc(sizeof(nodo));
    novo->prox = s->topo;
    s->topo->item = x;
    s->topo = novo;
}
```

Exercício 2

Pilha

Algoritmo 6: Desempilha

Entrada: Pilha S.
Saída: Item desempilhado.

```
1 início
2     se (PilhaVazia(S)) então
3         Imprima "Erro underflow: pilha vazia."
4     senão
5         aux ← S.topo
6         S.topo ← aux.prox
7         item ← aux.prox.item
8         DESALOCA_NODO(aux)
9         retorna item
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  int desempilha(pilha *s) {
2      int item = -1;
3      if (pilhaVazia(s))
4          printf("Underflow: pilha vazia.\n");
5      else {
6          nodo *aux = s->topo;
7          s->topo = aux->prox;
8          item = aux->prox->item;
9          free(aux);
10     }
11     return item;
12 }
```

Exercício 2

Pilha

- Ainda falta uma função: aquela que apaga todos os nodos de uma pilha.
- Vamos implementá-la a seguir.

Exercício 2

Pilha

Algoritmo 7: ApagaPilha

Entrada: Pilha S.

```
1 início
2     enquanto (NOT(PilhaVazia(S))) faça
3         // Ignora o elemento desempilhado.
4         Desempilha(S)
5     // Apaga o nodo cabeça.
6     DESALOCA_NODO(S.topo)
7     S.topo ← NULL
8     S.fundo ← NULL
```

Exercício 2

Pilha

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void apagaPilha(pilha *s) {
2     while (!pilhaVazia(s))
3         desempilha(s);
4     free(s->topo);
5     s->topo = NULL;
6     s->fundo = NULL;
7 }
```

Exercício 2

Pilha

Após implementar essas funções, faça o teste, empilhando e desempilhando alguns valores.

Exercício 2

Pilha

Em um arquivo chamado executa_pilha.c, insira o código a seguir:

```c
#include "pilha.h"
int main() {
    int i;
    pilha *s = malloc(sizeof(pilha));
    criaPilhaVazia(s);
    for(i=0;i<10;i++) {
        empilha(s,i);
        printf("Empilhou %d\n",i);
    }
    while(!pilhaVazia(s)) {
        i = desempilha(s);
        printf("Desempilhou %d\n",i);
    }
    apagaPilha(s);
    free(s);
}
```

Roteiro da Aula

Exercício 3

Fila

Neste exercício, vamos implementar a estrutura de dados **Fila**:

- Nessa estrutura existem dois ponteiros:
  - um aponta para o **início** da Fila.
  - o outro que aponta para o **fim** da Fila.
- No início da Fila existe um nodo vazio (denominado nodo **cabeça**).
- O ponteiro início sempre aponta para o nodo cabeça.
- Quando a Fila está vazia, início e fim apontam para o nodo cabeça.

Exercício 3

Fila

Considere o pseudo-código a seguir:

Algoritmo 8: Fila

```
1 início
2     registro {
3         Ponteiro Nodo: início, fim;
4     } Fila;
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 #include "nodo.h"
2 typedef struct {
3     nodo *inicio, *fim;
4 } fila;
```

Vamos chamar o arquivo que contém esse código de `fila.h`.

Exercício 3

Fila

Implemente uma fila utilizando alocação dinâmica. As operações básicas são:

1. CriaFilaVazia(Q)
2. FilaVazia(Q)
3. Enfileira(Q,x)
4. Desenfileira(Q)
5. ApagaFila(Q)

Exercício 3

Fila

Vamos adicionar o protótipo de cada uma das funções no arquivo `fila.h`.

```
1 void criaFilaVazia(fila *q);
2 int filaVazia(fila *q);
3 void enfileira(fila *q, int x);
4 int desenfileira(fila *q);
5 void apagaFila(fila *q);
```

Agora, vamos implementar cada uma dessas funções.

Exercício 3

Fila

- Primeiramente, de forma similar ao que fizemos com o TAD pilha, vamos criar um arquivo chamado `fila.c` que deve conter a implementação dessas funções.
- Para evitar que o compilador não encontre determinada função, vamos incluir o protótipo das funções, inserindo a linha de código a seguir no início do arquivo `fila.c`:

```
1 #include "fila.h"
```

- A seguir, é apresentado a implementação das funções, que devem ser incluídas no arquivo `fila.c`.

Exercício 3

Fila

Algoritmo 9: CriaFilaVazia

Entrada: Fila Q.

```
1 início
2     Q.inicio ← ALOCA_NODO()
3     Q.fim ← Q.inicio
4     Q.inicio.prox ← NULL
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void criaFilaVazia(fila *q) {
2     q->inicio = malloc(sizeof(nodo));
3     q->fim = q->inicio;
4     q->inicio->prox = NULL;
5 }
```

Exercício 3

Fila

Algoritmo 10: FilaVazia

Entrada: Fila Q.
Saída: Booleano (V ou F) indicando se Q está vazia.

```
1 início
2     se (Q.inicio = Q.fim) então
3         retorna VERDADEIRO
4     senão
5         retorna FALSO
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 int filaVazia(fila *q) {
2     return q->inicio == q->fim;
3 }
```

Exercício 3

Fila

Algoritmo 11: Enfileira

Entrada: Fila Q, item x.

```
1 início
2     Q.fim.prox ← ALOCA_NODO()
3     Q.fim ← Q.fim.prox
4     Q.fim.item ← x
5     Q.fim.prox ← NULL
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void enfileira(fila *q, int x) {
2     q->fim->prox = malloc(sizeof(nodo));
3     q->fim = q->fim->prox;
4     q->fim->item = x;
5     q->fim->prox = NULL;
6 }
```

Exercício 3

Fila

Algoritmo 12: Desenfileira

Entrada: Fila Q.
Saída: Item desenfileirado.

```
1  início
2      se ( FilaVazia(Q) ) então
3          Imprima "Erro underflow: fila esta vazia."
4      senão
5          aux ← Q.inicio
6          Q.inicio ← Q.inicio.prox
7          item ← Q.inicio.item
8          aux.prox ← NULL
9          DESALOCA_NODO(aux)
10         retorna item
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  int desenfileira(fila *q) {
2      int item = -1;
3      if (filaVazia(q)) {
4          printf("Underflow: fila vazia.\n");
5      }
6      else {
7          nodo *aux = q->inicio;
8          q->inicio = q->inicio->prox;
9          item = q->inicio->item;
10         aux->prox = NULL;
11         free(aux);
12     }
13     return item;
14 }
```

Exercício 3

Fila

- Assim como no TAD pilha, devemos implementar a função que apaga todos os nodos de uma fila.
- Vamos implementá-la a seguir.

Exercício 3

Fila

Algoritmo 13: ApagaFila

Entrada: Fila Q.

```
1 início
2     enquanto (NOT(FilaVazia(Q))) faça
3         // Ignora o elemento desenfileirado.
4         Desenfila(Q)
5     // Apaga o nodo cabeça.
6     DESALOCA_NODO(Q.inicio)
7     Q.inicio ← NULL
8     Q.fim ← NULL
```

Exercício 3

Fila

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void apagaFila(fila *q) {
2     while(!filaVazia(q))
3         desenfileira(q);
4     free(q->inicio);
5     q->inicio = NULL;
6     q->fim = NULL;
7 }
```

Exercício 2

Fila

Após implementar essas funções, faça o teste, enfileirando e desenfileirando alguns valores.

Exercício 2

Fila

Em um arquivo chamado executa_fila.c, insira o código a seguir:

```c
#include "fila.h"
int main() {
    int i;
    fila *q = malloc(sizeof(fila));
    criaFilaVazia(q);
    for (i=0;i<10;i++) {
        enfileira(q,i);
        printf("Enfileirou %d\n",i);
    }
    while (!filaVazia(q)) {
        i = desenfileira(q);
        printf("Desenfileirou %d\n",i);
    }
    apagaFila(q);
    free(q);
}
```

Pilha, Fila
Algoritmos e Estrutura de Dados
Pilha e Fila
(Alocação Dinâmica)

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia