

Algoritmos e Estrutura de Dados I - Aula 2

Michel Pires da Silva
michel@cefetmg.br

Departamento de Informática, Gestão e Design
DIGDDV

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG

14 de agosto de 2020

Tipo de Dados Lista - Apontador

O que muda a partir de agora?

- Agora, nosso tipo Apontador não faz mais referência a um valor diretamente, mas sim, a um endereço de memória onde o valor está armazenado
- A utilização de Ponteiros torna as estruturas dinâmicas, ou seja, sem limitação a primeiro momento.
- No caso das listas, a remoção é beneficiada com a eliminação da necessidade de movimentação dos itens

Tipo de Dados Lista - Apontador

A figura abaixo define a estrutura básica para a lista por apontador



Cabeça da lista.
Sempre vazia

Essa estrutura é a
célula da lista
(item + apontador)

Tipo de Dados Lista - Apontador

Observações gerais

- Em Pascal representamos os **Apontadores** com um ^ (circunflêxo) antes da declaração de **Tipo** da variável.
- Assim, tem-se:
 - ▶ **type** Apontador = ^ Celula; Nesse contexto, *Apontador* será um ponteiro do tipo *Celula*
 - ▶ Agora, teremos uma variável dentro do tipo *Celula* que aponta para outra *Celula*, criando a ligação necessária dentro da memória

Tipo de Dados Lista - Apontador

A estrutura básica para este novo modelo de declaração:

type

Apontador = ^Celula;

TipoItem = **record**

Chave: TipoChave;

{ *outros componentes* }

end;

Celula = **record**

Item: TipoItem;

Prox: Apontador;


end;

TipoLista = **record**

Primeiro: Apontador;

Ultimo : Apontador;

end;



*Apontadores para
Primeiro e Ultimo
são do tipo **Célula***

Tipo de Dados Lista - Apontador

Tratamento das opções de vazio ...

```
procedure FLVazia (var Lista : TipoLista);  
begin  
    new(Lista.Primeiro); // Diretriz new cria espaços de memória  
    Lista.Ultimo := Lista.Primeiro;  
    Lista.Primeiro^.Prox := nil;  
end
```

```
function Vazia (var Lista : TipoLista):boolean;  
begin  
    Vazia := Lista.Primeiro = Lista.Ultimo;  
end
```

Tipo de Dados Lista - Apontador

A estrutura do método de Inserção ...

```
procedure Insere (x : TipoItem; var Lista : TipoLista);  
begin  
    new(Lista.Ultimo^.Prox);  
    Lista.Ultimo := Lista.Ultimo^.Prox;  
    Lista.Ultimo^.Item := x;  
    Lista.Ultimo^.Prox := nil;  
end
```

Observações

- 1 A diretriz **new** se encarrega de criar um novo espaço e liga-lo ao último já existente - função exclusiva do Pascal
- 2 O apontador de *Ultimo* é direcionado para a nova posição
- 3 A nova última posição recebe o item e faz seu *Prox* apontar para **nil**, isso garante o final da fila.

Tipo de Dados Lista - Apontador

```
procedure Retira (p : Apontador; var x : TipoItem; var Lista : TipoLista);  
var aux : Apontador;  
begin  
    if (Vazia(Lista)) or (p = nil) or (p^.Prox = nil) then  
        | writeln( 'Erro: Lista vazia ou posição não existe');  
    end  
    else  
        aux := p^.Prox; // O elemento removido sempre será o Prox de  
        | p  
        x := aux^.Item;  
        p^.Prox := aux^.Prox;  
        if p^.Prox = nil then  
            | Lista.Ultimo := p;  
        end  
        dispose(aux); // Libera o espaço de memória  
    end  
end
```


Tipo de Dados Lista - Apontador

```
procedure Imprime (var Lista : TipoLista);  
var aux : Apontador;  
begin  
    aux := Lista.Primeiro^.Prox; // Lembre-se da cabeça  
    vazia  
    while aux <> nil do  
        writeln(aux^.item.chave);  
        aux := aux^.Prox; // Caminha para o próximo  
    end  
end
```

Observação

- 1 Observe que o procedimento para imprimir os elementos tem seu delimitador na diretriz **nil**
- 2 O apontador *Lista.Primeiro* sempre armazena a cabeça da lista cujo item não retem informação

Tipo de Dados Lista - Apontador

Vantagens e Desvantagens do tipo Lista por apontador

Vantagens

- Permite inserir ou retirar itens no meio da lista a um custo constante
 - ▶ Função importante, principalmente, para listas que apresentam como característica a manutenção de ordenação
- Boa opção para problemas que não se sabe a priori o tamanho exato do conjunto de dados

Desvantagens

- Utilização extra de memória para armazenar os apontadores

Tipo de Dados Lista - Apontador

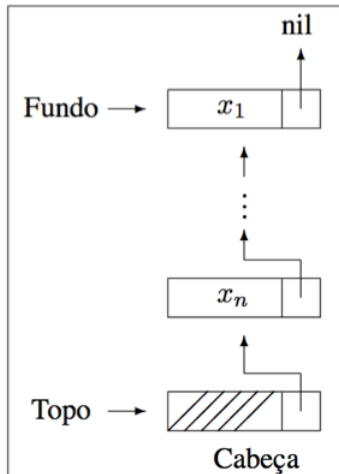
Exercício de Fixação:

Utilizando o que foi visto até agora tente elaborar soluções para as seguintes questões:

- Como é possível fazer com que a estrutura Lista apresentada se torne uma *Lista Simplesmente Encadeada Circular*
- Como você faria para inverter a lista por apontador sem trocar o conteúdo de cada célula de lugar ? Esse processo exigiria modificações na estrutura básica atual ?

Tipo de Dados Pilha - Apontador

- A *célula cabeça* também é definida para a Pilha, no entanto, ela se posiciona sempre no topo da estrutura.
- Note que a manipulação da estrutura será realizada a partir de uma célula vazia, logo será preciso movimentar o ponteiro antes de executar as operações de empilhar e desempilhar.
- A estrutura agora apresenta uma variável para computar o tamanho. Isso evita a contagem repetitiva



Tipo de Dados Pilha - Apontador

A estrutura básica para construirmos a pilha é definida como:

```
type
  Apontador = ^ Celula; /* Apontador de memória */
  TipoChave = Integer;

  TipoItem = Record
    Chave : TipoChave;
end;

  Celula = Record
    Item : TipoItem;
    Prox : Apontador; /* Aponta para a próxima célula */
end;

  TipoPilha = Record
    Fundo : Apontador;
    Topo : Apontador;
    Tamanho : Integer; /* Contabiliza o tamanho da pilha
                        */
end;
```

Tipo de Dados Pilha - Apontador

Tratamento das opções de vazio ...

```
procedure FPVazia (var Pilha : TipoPilha);  
begin  
    new(Pilha.Topo); // Diretriz new cria espaços de memória  
    Pilha.Fundo := Pilha.Topo;  
    Pilha.Topo^.Prox := nil;  
    Pilha.Tamanho := 0;  
end
```

```
function Vazia (var Pilha : TipoPilha):boolean;  
begin  
    Vazia := Pilha.Topo = Pilha.Fundo;  
end
```

Tipo de Dados Pilha - Apontador

Tratando as particularidades do método empilhar

```
procedure Empilha (x : TipoItem; var Pilha : TipoPilha);  
var Aux : Apontador;  
begin  
    new(Aux);  
    Pilha.Topo^.Item := x;  
    Aux^.Prox := Pilha.Topo;  
    Pilha.Topo := Aux;  
    Pilha.Tamanho := Pilha.Tamanho + 1;  
end
```

Tipo de Dados Pilha - Apontador

O procedimento desempilha e suas funções:

```
procedure Desempilha (var Item : TipoItem; var Pilha : TipoPilha);  
var Aux : Apontador;  
begin  
    if Vazia(Pilha) then  
        | writeln( 'Erro: Pilha vazia');  
    end  
    else  
        | Aux := Pilha.Topo;  
        | Pilha.Topo := Aux^.Prox;  
        | Item := Aux^.Prox^.Item; // Ou utilizar Pilha.Topo^.Item  
        | dispose(Aux);  
        | Pilha.Tamanho := Pilha.Tamanho - 1;  
    end  
end
```

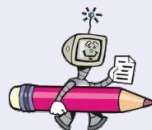

Tipo de Dados Pilha - Apontador

A função para retornar o tamanho da Pilha:

```
function Tamanho ( var Pilha : TipoPilha ):Integer;  
begin  
  | Tamanho := Pilha.Tamanho;  
end
```

Tarefa ...

Como ficaria uma função ou procedimento para imprimir todos os elementos da pilha sem perder o seu conteúdo?



Tipo de Dados Fila - Apontador

Observações gerais:

- Nessa estrutura, a célula cabeça é utilizada para facilitar as operações de enfileirar e desenfileirar
- Quando a fila encontra-se vazia ambos os apontadores, frente e tras apontarão para o mesmo endereço de memória
- Nessa estrutura, para desenfileirar um item, basta deslocar a célula cabeça para o *Prox*

Tipo de Dados Fila - Apontador

A estrutura básica para construirmos a fila é definida como:

```
type
  Apontador = ^ Celula; /* Apontador de memória */
  TipoChave = Integer;

  TipoItem = Record
    Chave : TipoChave;
end;

  Celula = Record
    Item : TipoItem;
    Prox : Apontador; /* Aponta para a próxima célula */
end;

  TipoFila = Record
    Frente : Apontador;
    Tras : Apontador;
end;
```

Tipo de Dados Fila - Apontador

Os procedimentos / funções para tratamendo do vazio:

```
procedure FFVazia ( var Fila : TipoFila );  
begin  
    | new (Fila.Frente);  
    | Fila.Tras := Fila.Frente;  
    | Fila.Frente^.Prox := nil;  
end
```

```
function Vazia ( var Fila : TipoFila ): boolean;  
begin  
    | Vazia := Fila.Frente = Fila.Tras;  
end
```

Tipo de Dados Fila - Apontador

Para as inserções, tem-se o método enfileirar abaixo:

```
procedure Enfileira ( x : TipoItem; var Fila : TipoFila );  
begin  
    new ( Fila.Tras^.Prox);  
    Fila.Tras := Fila.Tras^.Prox;  
    Fila.Tras^.Item := x;  
    Fila.Tras^.Prox := nil;  
end
```

Observação

Note que o único ponteiro de manipulação nesse caso é o **Fila.Tras**

Tipo de Dados Fila - Apontador

Para as remoções, tem-se o método desenfileira abaixo:

```
procedure Desenfileira ( var item : TipoItem; var Fila : TipoFila );  
var aux : Apontador;  
begin  
    if Vazia(Fila) then  
        | writeln ('Erro: Fila está vazia');  
    end  
    else  
        | aux := Fila.Frente;  
        | Fila.Frente := Fila.Frente^.Prox;  
        | item := Fila.Frente^.item;  
        | dispose(aux);  
    end  
end
```

Tipo de Dados Fila - Apontador

Exercício de fixação:

O caminhamento em matrizes é realizado por meio de dois ponteiros i, j , os quais definem a posição exata do elemento na estrutura. Sua tarefa é utilizar o conceito de fila para simular uma estrutura de matriz. Utilize dessa nova estrutura para responder as seguintes perguntas:

- a)- O tempo de acesso da nova estrutura é quanto mais caro ou barato que a padrão por matriz
- b)- Seria possível elaborar uma estrutura diferente para gerar melhor desempenho do que a fila simplesmente encadeada. Se há como, explique como fazer.