

Algoritmos e Estrutura de Dados I

Estruturas Básicas

Michel Pires da Silva
michel@cefetmg.br

Departamento de Computação
DECOMDV

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG

Sumário

- 1 Um problema vs Várias soluções
- 2 Estruturas de Dados Básicas por Arranjo
 - Tipo de Dados Lista
 - Tipo de Dados Pilha
 - Tipo de Dados Fila
- 3 Estruturas de Dados Básicas por Apontador
 - Tipo de Dados Lista - Apontador
 - Tipo de Dados Pilha - Apontador
 - Tipo de Dados Fila - Apontador

Labirinto

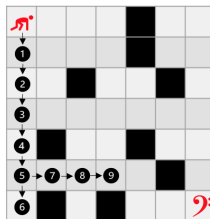
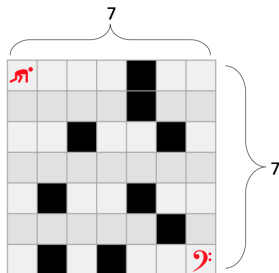


Figura 1: Busca em profundidade controlada por pilha

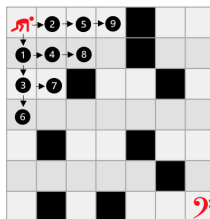


Figura 2: Busca em largura controlada por fila

Regras:

1. O jogo deve ser elaborado utilizando um arquivo de configuração. Nele deve conter: (a) tamanho da matriz; (b) posições das paredes e; (c) tipo de busca.
2. O Jogo termina assim que for atingido o alvo.
3. É preciso imprimir o caminho em tela sob uma representação de matriz. Então, imprima conforme exercício de matriz já realizado.

Pergunta:

1. Para diferentes tamanhos de matriz e posicionamento de paredes, há predominância de um dos dois algoritmos em termos de casas caminhas e tempo de execução?
2. Um dos dois algoritmos consegue encontrar o melhor caminho, ou seja, o com menor número de passos?

Tipo de Dados Lista

A maneira mais simples de interligar elementos de um conjunto é por meio de uma lista

- Listas definem uma estrutura composta de operações de **inserção**, **remoção** e **localização de elementos**.

Começamos pela estrutura lista porque é a estrutura mais flexível na manipulação de massas de dados.

- Listas são **adequadas** para aplicações onde não é possível prever com antecedência a demanda por memória e, muito menos, o comportamento dos dados.

Tipo de Dados Lista

As listas são úteis em aplicações como, manipulação simbólica, gerência de memória, simulação e compiladores.

Veremos hoje as *listas lineares* ...

- Representada por uma sequência de zero ou mais itens $\langle x_1, x_2, \dots, x_n \rangle$, sendo que x_i é de determinado tipo e n representa o tamanho da lista
- Para a criação da TAD lista é necessário um conjunto de operações, sendo esse determinado pela aplicação e/ou problema a ser resolvido

Tipo de Dados Lista

Usualmente, utilizamos um conjunto padrão de operações de manipulação formado por:

- Método para tornar a lista vazia
- Método para inserir um elemento após o i -ésimo elemento
- Método para retirar o i -ésimo elemento da lista
- Método para localizar o i -ésimo elemento da lista
- Método para realizar a junção de duas ou mais listas
- Método para dividir a lista em duas ou mais listas
- Método para gerar uma cópia da lista
- Método para ordenar a lista
- Método de pesquisa

Tipo de Dados Lista

- Existem várias estruturas de dados que podem ser utilizadas para representar listas lineares, as que utilizaremos englobam: structs e ponteiros
- A primeira estrutura que será desenvolvida leva em consideração o tipo **struct**

		Itens
Primeiro	0	x_1
	1	x_2
		\vdots
Último	$n - 1$	x_n
		\vdots
MaxTam		

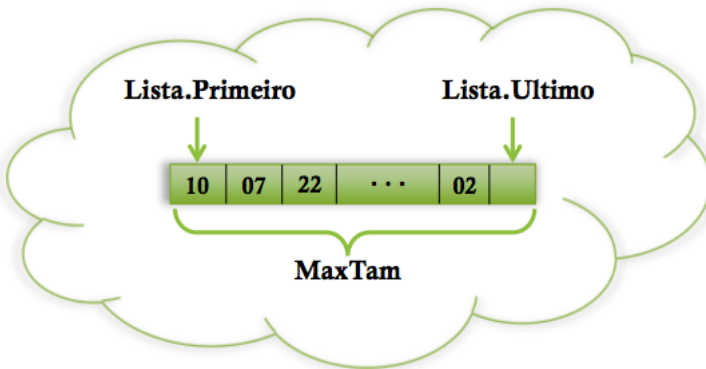
Tipo de Dados Lista

As listas lineares por arranjo apresentam as seguintes características:

- Os itens são armazenados em um **array** de tamanho suficiente para conter todo o conjunto de dados
- O campo **primeiro** é necessário para adaptarmos futuramente ao tipo por *apontador*
- O campo **último** aponta para a posição seguinte a do último elemento da lista
- O i -ésimo item da lista está armazenado na i -ésima posição do **array**, $1 \leq i \leq \text{último}$
- A constante *MaxTam* define o tamanho máximo permitido para o armazenamento de elementos na lista

Tipo de Dados Lista

A estrutura base para a nossa TAD Lista pode ser observada abaixo:



Tipo de Dados Lista

const

InícioArranjo = 1;
MaxTam = 1000;

type

TipoChave = **integer**;
Apontador = **integer**;
TipoItem = **record**
 Chave: TipoChave;
 { *outros componentes* }

end;

TipoLista = **record**

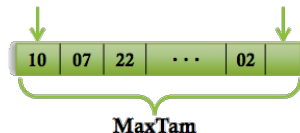
Item : **array** [1..MaxTam] of TipoItem;
Primeiro : Apontador;
Ultimo : Apontador

end;

Estrutura
necessária para a
manipulação da
lista por arranjo

Lista.Primeiro

Lista.Ultimo



Tipo de Dados Lista

Procedimento / função para inicializar a lista e verificar se a mesma está vazia ou não (**True** ou **False**)

```
procedure FLVazia (var Lista : TipoLista);
```

```
begin
```

```
    Lista.Primeiro := InicioArranjo;
```

```
    Lista.Ultimo := Lista.Primeiro;
```

```
end;
```

```
function Vazia (var Lista : TipoLista): boolean;
```

```
begin
```

```
    Vazia := Lista.Primeiro = Lista.Ultimo;
```

```
end;
```

Lista.Primeiro

Lista.Ultimo

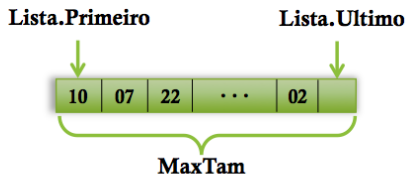


MaxTam

Tipo de Dados Lista

Procedimento responsável por gerar as inserções na lista e avançar o ponteiro de último para a próxima posição vazia

```
procedure Insere (x: TipoItem; var Lista: TipoLista);  
begin  
  if Lista.Ultimo > MaxTam  
  then writeln('Lista esta cheia')  
  else begin  
    Lista.Item[Lista.Ultimo] := x;  
    Lista.Ultimo := Lista.Ultimo + 1;  
  end;  
end;
```



Tipo de Dados Lista

Procedimento responsável por gerar as remoções na lista e retroceder o ponteiro de último para a próxima posição vazia

```
procedure Retira (p:Apontador; var Lista:TipoLista;  
                 var Item:TipoItem);
```

```
var Aux: integer;
```

```
begin
```

```
  if Vazia (Lista) or (p >= Lista.Ultimo)
```

```
  then writeln ( 'Erro: Posicao nao existe' )
```

```
  else begin
```

```
    Item := Lista.Item[p];
```

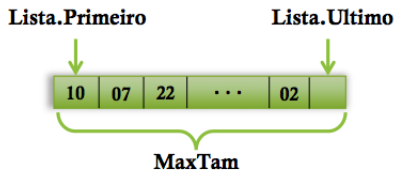
```
    Lista.Ultimo := Lista.Ultimo - 1;
```

```
    for Aux := p to Lista.Ultimo - 1 do
```

```
      Lista.Item[Aux] := Lista.Item[Aux+1];
```

```
    end;
```

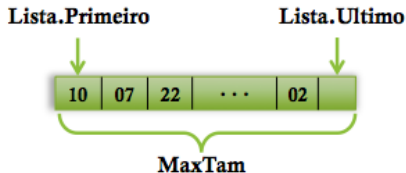
```
end;
```



Tipo de Dados Lista

Procedimento responsável por imprimir a lista

```
procedure Imprime (var Lista: TipoLista);  
var Aux: integer;  
begin  
    for Aux := Lista.Primeiro to Lista.Ultimo - 1 do  
        writeln (Lista.Item[Aux].Chave);  
end;
```



Tipo de Dados Lista

Quais as vantagens e desvantagens da estrutura apresentada ?

Vantagens

Economia de memória porque os apontadores ficam implícitos na estrutura.

Desvantagens

- Custo alto para a remoção de elementos do início da lista já que é necessário que o procedimento "empurre" todos os elementos restantes para uma posição a frente.
- A definição do **MaxTam** em tempo de compilação gera uma limitação em termos de tamanho máximo que a lista pode ter.

Tipo de Dados Lista

Exercício 01: Faça um diagrama para representar uma inserção, remoção e pesquisa em uma lista linear.

Exercício 02: Crie uma lista linear que consiga armazenar um conjunto de 100 nomes quaisquer, os quais serão salvos de forma aleatória. Feito a estrutura crie as seguintes ações:

- Uma função que consiga identificar replicações na lista, ou seja, nomes iguais. Remova todas as réplicas sem mover os "ponteiro".
- Como você gerenciaria os espaços em branco para novas inserções? Qual seria o custo dessas novas inserções?

Exercício 03: O problema da máxima cadeia. Elabore um programa que receba uma cadeias de DNA. Cada posição da cadeia deve conter um códon, ou seja, uma triade de nucleotídeos \rightarrow T, A, G, C. Feito isso, leia de um arquivo uma sequência de nucleotídios (i.e., ACGTGGCTCTCTAACGTACGTACGTACGGGGTTATATTCGAT) e tente identificar a maior cadeia da lista que se relaciona a essa entrada.

Tipo de Dados Lista

Exercício 04: Em uma lista A temos um conjunto de elementos inteiros positivos ou não $\langle a_1, a_2, \dots, a_n \rangle$. Elabore uma função que consiga encontrar neste conjunto a máxima soma.

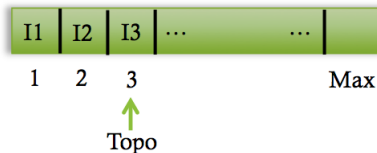
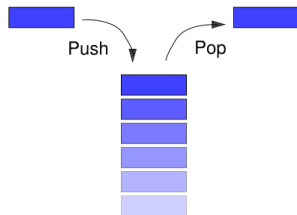
Exercício 05: Faça a especificação de um sistema de controle de reservas de um clube que aluga quadras poliesportivas usando Lista. Nesse modelo de reservas, as quadras tem limite de hora para serem utilizadas, uma hora por reserva.

Desafio: Em sala chegamos a discutir a possibilidade da lista ser circular e "infinita". Como você implementaria tal modelo de estrutura? Como controlar os ponteiros de primeiro e último? Faça um diagrama para representar essa estrutura e funções para descrever sua solução.

Tipo de Dados Pilha

Conceitos Introdutórios:

- É uma lista linear em que todas as inserções, remoções e, geralmente, todos os acessos são feitos apenas em um extremo.
- Nessa estrutura os itens são colocados uns sobre os outros. O item inserido mais recentemente estará no topo e o mais antigo no fundo da estrutura.
- O modelo intuitivo é o de um monte de pratos em uma prateleira, sendo conveniente tirar ou inserir pratos no topo do monte.



Tipo de Dados Pilha

Propriedades e Aplicações:

- O último elemento inserido é o primeiro que pode ser retirado - LIFO (*last in first out*).
- É ideal para processamento de estruturas *aninhadas* de profundidade imprevisível, controle de tarefas a fazer posteriormente, quando há recursão e sintaxe de expressões aritméticas.
- Garante a execução da parte mais interna primeiro.

Tipo de Dados Pilha

Conjunto de Operações Básicas:

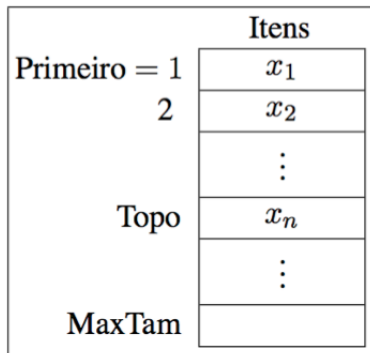
- **FPVazia(Pilha):** Esvazia a pilha para iniciar as execuções
- **Vazia(Pilha):** Retorna TRUE se a pilha estiver vazia; Caso contrário, retorna FALSE.
- **Empilha(X, Pilha):** Insere o item X no topo da pilha
- **Desempilha(X, Pilha):** Retira o item do topo da pilha
- **Tamanho(Pilha):** Retorna o tamanho da pilha

Tipo de Dados Pilha

O modelo por Arranjo:

Os elementos da pilha se encontram em posições contínuas de memória no modelo por arranjo


Nesse modelo, um cursor chamado **topo** é utilizado para identificar o item que está no topo da pilha.



Tipo de Dados Pilha

Estrutura Básica:

```
const MaxTam = 1000;  
type  
  TipoChave = integer;  
  Apontador = integer;  
  TipoItem  = record  
    Chave: TipoChave;  
    { outros componentes }  
  end;  
  TipoPilha = record  
    Item: array [1..MaxTam] of TipoItem;  
    Topo: Apontador;  
  end;
```



Note as
semelhanças
com o tipo
lista

Tipo de Dados Pilha

Operações para o tratamento do vazio:

```
procedure FPVazia (var Pilha: TipoPilha);  
begin  
    Pilha.Topo := 0;  
end;
```

```
function Vazia (var Pilha: TipoPilha): boolean;  
begin  
    Vazia := Pilha.Topo = 0;  
end;
```

Tipo de Dados Pilha

Operação para empilhar - *Push*:

```
procedure Empilha (x: TipoItem; var Pilha: TipoPilha);  
begin  
    if Pilha.Topo = MaxTam  
    then writeln ( 'Erro: pilha esta cheia' )  
    else begin  
        Pilha.Topo := Pilha.Topo + 1;  
        Pilha.Item[Pilha.Topo] := x;  
    end;  
end;
```


Tipo de Dados Pilha

Operação para desempilhar - *Pop*:

```
procedure Desempilha (var Pilha: TipoPilha; var Item: TipoItem);  
begin  
    if Vazia (Pilha)  
    then writeIn ( 'Erro: pilha esta vazia' )  
    else begin  
        Item := Pilha.Item[Pilha.Topo];  
        Pilha.Topo := Pilha.Topo - 1;  
    end;  
end;
```

Tipo de Dados Pilha

Operação para avaliar o tamanho da pilha:

```
function Tamanho (Pilha : TipoPilha) : integer ;  
begin  
    Tamanho := Pilha.Topo;  
end;
```

Observação ...

Essa função facilita, muitas vezes, algumas verificações que são necessárias ao trabalharmos com o tipo pilha.

Exercício

Exercício de fixação:

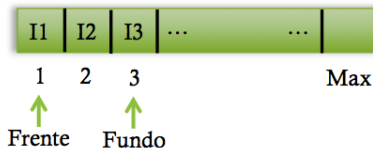
Um matemático está preocupado com a forma que seus alunos estão avaliando algumas fórmulas matemáticas. Eles estão esquecendo de validar o número de operandos e operadores da mesma antes de prosseguir com os cálculos.

Vejamos um exemplo: $2 + 2$ (operação correta / completa); $2 + 2 -$ (falta de um operando); $2 2$ (falta de um operador). Sua função é utilizar o tipo pilha para criar um procedimento que consiga validar as operações e dizer se estão ou não corretas.

Tipo de Dados Fila

Conceitos Introdutórios:

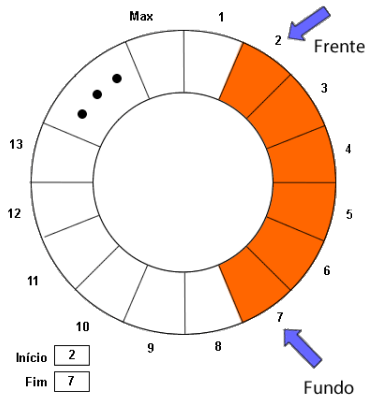
- É uma lista em que todas as **inserções** são realizadas no **final** e todas as **remoções** são feitas no **início**.
- O modelo fila está presente em muitos lugares reais, tais como, bancos, lotéricas, etc ...
- Sua estrutura e operações criam um modelo chamado FIFO - *first in first out*
- São utilizadas quando há necessidade de processar os itens na ordem de chegada.



Tipo de Dados Fila

A estrutura de dados Fila faz uso de dois ponteiros: **frente** e **fundo**

Problema: Como ambos caminham até MaxTam precisamos visualizar a lista como se fosse circular.



Visão circular

Para gerar a manipulação adequada é preciso deixar uma posição vazia quando a fila estiver cheia para não confundir a fila cheia com vazia.

Tipo de Dados Fila

Conjunto de Operações:

- **FFVazia(Fila):** Procedimento para a inicialização do tipo Fila
- **Vazia(Fila):** Procedimento que verifica se a fila está ou não vazia
- **Enfileirar(X, Fila):** Coloca-se o item X no final da fila
- **Desenfileirar(X, Fila):** Retira-se o primeiro item da fila

Tipo de Dados Fila

Estrutura Básica:

```
const MaxTam = 1000;

type
  TipoChave = integer;
  Apontador = integer;
  TipoItem  = record
    Chave: TipoChave;
    { outros componentes }
  end;

  TipoFila = record
    Item  : array [1..MaxTam] of TipoItem;
    Frente: Apontador;
    Tras  : Apontador;
  end;
```



Há semelhanças
com o tipo lista?

Tipo de Dados Fila

Operações para controle do vazio

```
procedure FFVazia (var Fila : TipoFila);  
begin  
    Fila.Frente := 1;  
    Fila.Tras    := Fila.Frente;  
end; { FFVazia }
```

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


Tipo de Dados Fila

Operação para enfileirar

```
procedure Enfileira (x: TipoItem; var Fila: TipoFila);  
begin  
    if Fila.Tras mod MaxTam + 1 = Fila.Frente  
    then writeln ('Erro: fila esta cheia')  
    else begin  
        Fila.Item[Fila.Tras] := x;  
        Fila.Tras := Fila.Tras mod MaxTam + 1;  
    end;  
  
end;
```

Tipo de Dados Fila

Operação para desenfileirar

```
procedure Desenfileira (var Fila: TipoFila;  
                        var Item: TipoItem);  
  
begin  
    if Vazia (Fila)  
    then writeLn ( 'Erro: fila esta vazia')  
    else begin  
        Item := Fila.Item[Fila.Frente];  
        Fila.Frente := Fila.Frente mod MaxTam + 1;  
    end;  
  
end;
```

Tipo de Dados Fila

Exercício de fixação:

Uma escola está com problemas para atender a demanda de alunos no setor de pagamentos. O que está acontecendo é que não se sabe ao certo quem chega primeiro e qual aluno deve ser atendido na frente do outro. Sua função é criar um procedimento que, dado uma fila vazia, possibilite seu preenchimento com 10 números de registro de alunos. Feito isso, imprima como resultado a sequência que os atendimentos devem ocorrer, removendo um item por vez mostrando-o na tela.

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

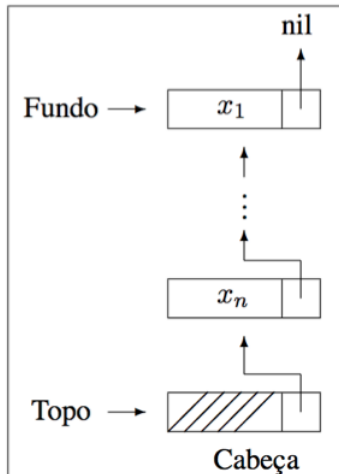
Exercício de Fixação:

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
```

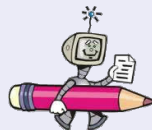
Tipo de Dados Pilha - Apontador

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 tratamento 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.

PERGUNTAS?

