



ESTRUTURAS DE DADOS

ROGÉRIO FERREIRA SGOTI

ESTRUTURAS DE DADOS

Instituição:	Faculdade de Tecnologia de Botucatu
Curso:	Análise e Desenvolvimento de Sistemas
Carga Horária:	04 h/a semanais – Total: 80 horas
Professor Responsável:	Rogério Ferreira Sgoti

CONTEÚDO PROGRAMÁTICO	
Semana	Tópicos
1ª	Apresentação da Disciplina: Objetivos, Ementa, Conteúdo Programático, Critério de Avaliação, Bibliografia. Revisão de Algoritmos (Rotinas).
2ª	Registros. Aplicações, Sintaxes e Exemplos. Exercícios.
3ª	Manipulação de registros. Exercícios. Correção de Exercícios de Registro.
4ª	Ordenação ou Classificação de dados. Conceitos. Algoritmo Bubble Sort. Simulação do algoritmo.
5ª	Algoritmos Insertion Sort e Selection Sort. Simulação dos algoritmos.
6ª	Pesquisa ou Busca de dados. Conceitos. Pesquisa Seqüencial e Pesquisa Binária. Algoritmos.
7ª	Cálculos de desempenho para algoritmos de pesquisa. Exemplos. Lista de Exercícios. Correção dos Exercícios.
8ª	Listas; Conceitos e Aplicações. Tipos de Listas. Lista Estática Seqüencial. Operações e Algoritmos.
9ª	Lista Estática Encadeada. Operações e Algoritmos.
10ª	Simulação dos algoritmos. Atendimento às dúvidas. 1ª Avaliação de Estruturas de Dados.
11ª	Listas Dinâmicas. Conceitos e Aplicações. Exemplos. Alocação Estática X Alocação Dinâmica de Memória.
12ª	Ponteiros. Conceitos. Notações. Exemplos. Exercícios.
13ª	Listas Dinâmicas. Operações e Algoritmos.
14ª	Exercícios. Correção dos exercícios. Atendimento às dúvidas.
15ª	Pilhas. Conceitos e Aplicações. Operações sobre Pilhas. Algoritmos.
16ª	Exercícios. Simulação dos exercícios.
17ª	Filas. Conceitos e Aplicações. Operações sobre Filas. Algoritmos.
18ª	Exercícios. Simulação dos exercícios.
19ª	Árvores. Conceitos e Aplicações. Operações sobre Árvores. Algoritmos. Exercícios. Simulação dos exercícios. Atendimento às dúvidas
20ª	2ª Avaliação de Estruturas de Dados. P3 de Estruturas de Dados.

1 – INTRODUÇÃO AO ESTUDO DAS ESTRUTURAS DE DADOS

Um programa de computador consiste em basicamente duas coisas: instruções e locais para se armazenarem dados. Desse modo, as linguagens de programação são “equipadas” com mecanismos para poder controlar as instruções e os dados. Esses mecanismos são denominados “estruturas” e todas as linguagens de programação contemporâneas contam com dois tipos de estruturas – as estruturas de controle de fluxo de execução (para instruções) e as estruturas de dados (para os dados).

✓ Estruturas de Controle de Fluxo de Execução

- Seqüencial;
- Condicionais;
- Iterativas;
- Chamadas a rotinas.

✓ Estruturas de Dados

- Primitivas (inteiro, real, lógico, caracter);
- Compostas
 - Homogêneas (vetores, matrizes);
 - Heterogêneas (registros, arquivos);
- Complexas
 - Estáticas (Listas Estáticas Seqüenciais, Listas Estáticas Encadeadas);
 - Dinâmicas (Listas Dinâmicas, Ponteiros, Pilhas, Filas, Árvores).

2 – REVISÃO DE TÓPICOS

- Modularização;
- Variáveis Locais e Globais;
- Passagem de Parâmetros.

Exemplos:

Algoritmos para cálculo do fatorial de um número n.

ALGORITMO FATORIAL_1;

VAR

valor : inteiro;

Procedimento FATORIAL (n: inteiro);

Var

i, fat : inteiro;

Inicio

fat \leftarrow 1;

para i \leftarrow ate n faca

fat \leftarrow fat * i;

fim para;

exiba ("Fatorial de ", n, " é: ", fat);

Fim;

INICIO

exiba ("Digite um número inteiro:");

leia (valor);

FATORIAL (valor);

exiba ("Tecle algo para encerrar...");

leia;

FIM.

ALGORITMO FATORIAL_2;

VAR

valor, result : inteiro;

Procedimento FATORIAL (n: inteiro; var fat: inteiro);

Var

i : inteiro;

Inicio

fat \leftarrow 1;

para i \leftarrow ate n faca

fat \leftarrow fat * i;

fim para;

Fim;

INICIO

exiba ("Digite um número inteiro:");

leia (valor);

FATORIAL (valor, result);

exiba ("Fatorial de ", valor, " é: ", result);

exiba ("Tecle algo para encerrar...");

leia;

FIM.

ALGORITMO FATORIAL_3;

VAR

valor : inteiro;

Funcao FATORIAL (n: inteiro): inteiro;

Var

i, fat : inteiro;

Inicio

fat \leftarrow 1;

para i \leftarrow ate n faca

fat \leftarrow fat + i;

fim para;

FATORIAL \leftarrow fat;

Fim;

INICIO

exiba ("Digite um número inteiro:");

leia (valor);

exiba("Fatorial de ", valor, "é: ", FATORIAL (valor));

exiba ("Tecle algo para encerrar...");

leia;

FIM.

EXERCÍCIOS

1) Escreva três algoritmos que recebam um número inteiro e exibam se esse número é primo ou não. Cada algoritmo com uma técnica diferente: procedimento com passagem de parâmetros por valor, procedimento com passagem de parâmetros por referência e por meio de função.

2) Escreva um algoritmo que receba 100 números quaisquer e armazene-os em um vetor. Calcular e exibir o maior número e o menor número.

3 – REGISTROS

As estruturas de dados utilizadas até o momento (vetores e matrizes), permitem o armazenamento de vários valores na mesma estrutura, porém, cada estrutura só pode ser definida para um único tipo de dados.

Quando se precisou utilizar dados de diferentes tipos para a solução de problemas, foi necessário o uso de diferentes estruturas.

Agora, é apresentada uma estrutura de dados em que é possível armazenar, na mesma estrutura, diversos valores de diferentes tipos de dados. Essa estrutura é chamada de Registro e, por sua característica, classificada como uma estrutura de dados heterogênea.

Um exemplo de registro:

BOLETIM ESCOLAR	
Aluno.....:	Fulano de Tal
1ª Nota.....:	8,5
2ª Nota.....:	9,0
3ª Nota.....:	7,5
Média.....:	7,5

Sintaxe:

TIPO

identificador_registro = REGISTRO

campo1: tipo de dado;

campo2: tipo de dado;

.

.

.

campoN: tipo de dado;

FIM;

VAR

identificador_variavel: identificador_registro;

Exemplo:

TIPO

cad_aluno = REGISTRO

nome: caracter;

nota1: real;

nota2: real;

nota3: real;

media: real;

FIM;

VAR

aluno : cad_aluno;

Referência:

identificador_variavel.campo

aluno.nome ← “João”;

leia (aluno.nota1);

exiba (aluno.media);

Exercício

Escreva um algoritmo que receba o nome e as 4 notas de aluno, calcule e exiba a média das notas e a situação (aprovado ou reprovado) baseado na média maior ou igual a 6,0.

Outro exemplo de registro:

BOLETIM ESCOLAR				
Aluno.....: Fulano de Tal				
Notas				
1	2	3	4	
Média.....: 7,5				

Exemplo:

```
TIPO
    bimestre = VETOR [1..4] de real;

    cad_aluno = REGISTRO
        nome: caracter;
        nota: bimestre;
        media: real;
    FIM;

VAR
    aluno : cad_aluno;
    i : inteiro;
```

Referência:

identificador_variavel.campo[i]

leia (aluno.nota[i]);

exiba (aluno.nota[i]);

Exercícios

1) Escreva um algoritmo para manipular a ocorrência de um único aluno e suas 4 notas bimestrais. Receba os dados do usuário e exiba o boletim conforme o último layout apresentado.

2) Idem ao anterior, porém manipular os dados para uma sala com 40 alunos. (E agora, como fica a estrutura de dados?).

TIPO

bimestre = VETOR [1..4] de real;

cad_aluno = REGISTRO

nome: caracter;

nota: bimestre;

media: real;

FIM;

aluno_vet = VETOR [1..40] de cad_aluno;

VAR

aluno : aluno_vet;

i, j : inteiro;

Referência:

identificador_variavel[i].campo[j]
--

leia (aluno[i].nome);

leia (aluno[i].nota [j]);

Exercícios

1) Considerando o cadastro de uma agenda pessoal de nomes, telefones e cidades, defina a estrutura de registros apropriada e construa um algoritmo para armazenar 50 contatos nessa agenda. Após o armazenamento, exibir um relatório com os dados de todos os contatos que residem em Botucatu.

2) Considerando o cadastro de uma agenda de prestadores de serviços contendo nome, código de área, telefone e cidade, defina a estrutura de registros apropriada e construa um algoritmo para armazenar 80 contatos nessa agenda. Após o armazenamento, exibir um relatório com os dados de todos os prestadores de serviço de uma determinada área de telefone. Essa informação deve ser solicitada para o usuário. Obs: o código de área deve ser um campo separado do número do telefone (para possibilitar a consulta prevista no algoritmo).

4 – ALGORITMOS DE ORDENAÇÃO OU CLASSIFICAÇÃO

✓ Algoritmo Bubble Sort (ou método da bolha)

Por ser simples e de fácil implementação o algoritmo Bubble Sort ou método da bolha está entre os mais conhecidos e difundidos métodos de ordenação de arranjos de dados. Mas não se trata do algoritmo mais eficiente. É estudado para desenvolvimento do raciocínio lógico e como introdução ao assunto de ordenação ou classificação de dados.

O princípio da dinâmica de funcionamento desse algoritmo é o teste e a troca de valores entre as posições consecutivas do arranjo, fazendo com que os valores mais altos (ou mais baixos) “borbulhem” para o final do arranjo (ou começo). Este algoritmo realiza $(n-1)$ iterações.

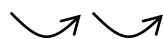
Exemplo:

1	2	3	4	5
7	9	8	5	3



1ª Iteração

1	2	3	4	5
7	8	5	3	9



2ª Iteração

1	2	3	4	5
7	5	3	8	9



3ª Iteração

1	2	3	4	5
5	3	7	8	9



4ª Iteração

1	2	3	4	5
3	5	7	8	9

Exercício

Escreva um algoritmo que armazene 100 valores num vetor e ordene-os crescentemente implementando o método bubblesort.

✓ Algoritmo Selection Sort

O algoritmo do método de ordenação Selection Sort seleciona o maior elemento do arranjo e troca-o com o elemento da última posição. A cada iteração desconsidera a “última” posição do vetor fazendo com que o arranjo fique “menor”. Esse algoritmo também realiza $(n-1)$ iterações para garantir que os elementos estejam ordenados.

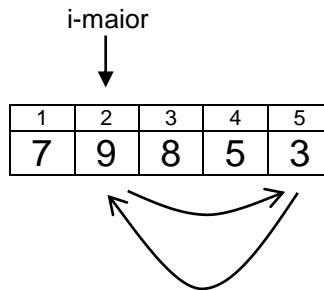
Dinâmica de funcionamento:

- Na primeira iteração o algoritmo encontra o maior elemento do vetor e troca este elemento com o elemento que está na n -ésima posição;
- Na segunda iteração despreza-se o n -ésimo elemento e encontra-se novamente o maior elemento (exceto o maior elemento já identificado na primeira iteração, este segundo maior é o maior entre os demais) e troca-se com o elemento que está na $(n-1)$ -ésima posição;
- Este processo se repete até que se tenha ordenado todos os elementos do arranjo.

Exemplo:

1	2	3	4	5
7	9	8	5	3

- Percorre-se o vetor na busca do maior elemento. Neste caso, o maior elemento é o 9, que está na posição 2. Troca-se então esse elemento com o elemento que está na última posição.



1ª Iteração

1	2	3	4	5
7	3	8	5	9

- Finalizada a primeira iteração inicia-se o processo novamente, selecionando o maior elemento e trocando-o com a penúltima posição. E assim sucessivamente...

Exercício

Escreva um algoritmo que armazene 100 valores num vetor e ordene-os crescentemente implementando o método selection sort.

✓ Algoritmo Insertion Sort

O algoritmo do método de ordenação Insertion Sort utiliza a técnica de indução algorítmica para resolver o problema de ordenação. Assim, sabendo-se resolver um problema menor, resolve-se um problema maior.

Dinâmica de funcionamento:

Considere o vetor V assim definido: $V = \{ V_1, V_2, V_3, \dots, V_{n-1}, V_n \}$.

Considere o vetor V' assim definido e já ordenado: $V' = \{ V_1 \}$.

A dinâmica consiste em comparar do segundo elemento em diante do vetor V com o elemento (sempre já ordenado) do vetor V' e inseri-los na posição correta do vetor V' .

Exemplo:

1	2	3	4	5
9	8	7	5	3


$\underbrace{\hspace{1.5cm}}_{V'} \quad \underbrace{\hspace{3.5cm}}_V$

Tem-se: $V' = 9$

$V = 8 \ 7 \ 5 \ 3$

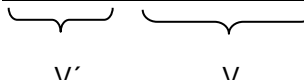
Inserir o elemento 8 no vetor já ordenado V' .

1	2	3	4	5
9	8	7	5	3



Para inserir o elemento 8 na posição correta do vetor V' , desloca-se o elemento 9 para uma posição a direita.


1	2	3	4	5
8	9	7	5	3



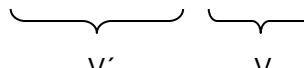
V' V

Assim, o processo se repete até que o vetor esteja ordenado.

1	2	3	4	5
8	9	7	5	3

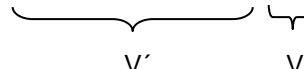


1	2	3	4	5
7	8	9	5	3



V' V

1	2	3	4	5
5	7	8	9	3



V' V

1	2	3	4	5
3	5	7	8	9

Exercício

Escreva um algoritmo que armazene 100 valores num vetor e ordene-os crescentemente implementando o método insertion sort.

5 – ALGORITMOS DE PESQUISA (OU BUSCA)

São dois os métodos de pesquisa ou busca por uma ocorrência de um elemento dentro de um arranjo de dados que são mais comumente usados. Geralmente é o usuário da aplicação quem fornece o valor do elemento que está procurando e que tem interesse na pesquisa. Esses dois métodos são: a pesquisa seqüencial e a pesquisa binária.

Qual desses tipos de pesquisa apresenta o melhor resultado depende exclusivamente da quantidade de operações realizadas para encontrar determinado elemento no arranjo, e isso é determinado por dois fatores: o número de elementos do arranjo e a quantidade de consultas que se faça no arranjo.

Após conhecer como funcionam os dois tipos de pesquisa, no próximo capítulo serão apresentados alguns cálculos sobre como calcular a quantidade de operações necessárias para encontrar (ou não) um elemento dentro do arranjo de dados e, com isso, poder decidir qual dos métodos é melhor aplicar.

✓ Pesquisa Seqüencial

O método de pesquisa seqüencial consiste em efetuar a busca da informação desejada a partir do primeiro elemento, percorrendo o vetor sequencialmente até o último elemento. Localizando a informação procurada em algum dos elementos, esta é apresentada ao usuário e encerra-se a busca.

Este método de pesquisa é, na média, um tanto lento. Porém, eficiente nos casos em que o vetor contenha seus elementos de forma desordenada.

Exercício: Escreva um algoritmo que implemente o método de pesquisa seqüencial.

✓ Pesquisa Binária

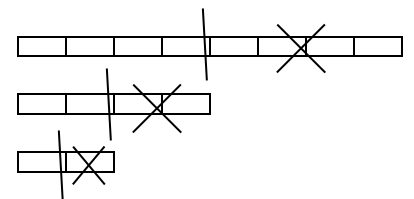
O método da pesquisa binária apresenta um mecanismo que, na média, se mostra mais rápido em comparação ao método da pesquisa seqüencial.

Uma diferença é que para aplicar a pesquisa binária o arranjo de dados deve estar obrigatoriamente ordenado previamente.

Depois de ordenado o vetor, este é dividido em duas partes e examinado se a informação a ser pesquisada se encontra na linha de divisão (meio) do vetor ou se está acima ou abaixo desta linha de divisão. Se estiver acima, toda a metade inferior é desprezada para verificação. Em seguida, se a informação não foi encontrada no meio (lembre-se de que o vetor já está ordenado!), novamente se divide esta metade do vetor em duas partes e se examina se a informação está no meio, acima ou abaixo da linha divisória.

O processo de divisão e descarte continua até que a informação seja encontrada ou até se acabar os elementos do arranjo.

Exercício: Escreva um algoritmo que implemente o método de pesquisa binária.



Exercício: Escreva um algoritmo que exiba um menu para o usuário com as opções: Tamanho do Vetor, Armazenar Valores, Ordenar Valores, Tipo de Pesquisa, Pesquisar e Sair; e implemente essas funções.

6 – COMPARATIVO ENTRE OS ALGORITMOS DE PESQUISA

✓ Cálculo do nº de Operações para Pesquisas Seqüencial e Binária

Considerações:

- Para a Ordenação do arranjo:
 - tem-se $(n - 1)$ iterações;
 - o vetor verificará $(n - 1)$ elementos;
 - logo, $(n - 1) \cdot (n - 1) = n^2 - 2n + 1$.
 - a qtd de operações é proporcional a n^2 desprezível
- Para a Pesquisa no arranjo:
 - $n = n^0$ de iterações por pesquisa ou n^0 de elementos do arranjo;
 - $m =$ qtd de pesquisas;
 - $op = n^0$ de operações realizadas.

✓ Pesquisa Seqüencial

$$Op = n \cdot m$$

✓ Pesquisa Binária

- Na ordenação: $op' = n^2$

- Na pesquisa:

$$\left. \begin{array}{lcl} 1^a \text{ iteração} & \rightarrow & n/2 = n/2^1 \\ 2^a \text{ iteração} & \rightarrow & n/4 = n/2^2 \\ 3^a \text{ iteração} & \rightarrow & n/8 = n/2^3 \\ \vdots & & \vdots \\ x^a \text{ iteração} & \rightarrow & n/2^x = \end{array} \right\} \log_2(n) = x \rightarrow 2^x = n$$

- Se forem realizadas m pesquisas, tem-se: $op'' = m \cdot \log_2(n)$

- Logo, o nº total de operações para a pesquisa binária é dado por $op = op' + op''$

- \therefore

$$Op = n^2 + m \cdot \log_2(n)$$

Exemplo: Arranjo com 8 elementos

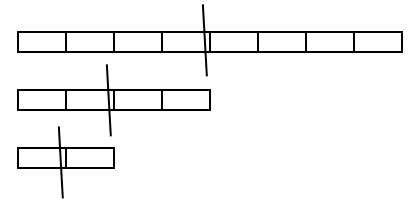
$$n = 8$$

$$\log_2 (8) =$$

$$2^x = 8$$

$$x = 3$$

\therefore São necessárias 3 iterações (operações)



Exercícios

1) Dado um vetor com 8 elementos, decidir e apontar qual o melhor tipo de pesquisa a ser aplicada no caso de necessidade de 1000 consultas nesse vetor.

2) Considere um arranjo com 8192 elementos. Qual o tipo de pesquisa é mais vantajoso para 1000 consultas?

3) Dado um vetor com 30 elementos, indicar qual o melhor método de pesquisa para uma necessidade de 3 consultas.

4) Determinado arranjo contém 3875 elementos e há necessidade de realizar 327 consultas. Nesse caso, qual método de pesquisa aplicar?

5) Dado um vetor com 1480 elementos e necessidade de 15 consultas qual tipo de pesquisa você implementaria?

7 – LISTAS

Uma Lista é uma estrutura de dados que armazena elementos de forma alinhada, ou seja, com elementos dispostos um após o outro, como em uma lista telefônica, um catálogo de peças, entre outros. Além disso, existem vários tipos de listas e o que os determinam é o modo e a política como são realizadas as operações sobre essas listas. As principais operações são, entre outras, inclusão de elementos na lista, acesso a um determinado elemento da lista, exclusão de um elemento, verificação de lista vazia, verificação de lista cheia, quantidade de elementos na lista.

A partir de agora praticamente todas as estruturas de dados que serão estudadas são derivadas de algum tipo de lista. As listas podem ser classificadas de várias formas, que dependem de alguns fatores como: o modo como é implementada a alocação de memória; a natureza de manipulação dos elementos da lista e ainda como fica a disposição dos elementos em relação à linearidade ou à hierarquia da estrutura.

Classificação e Tipos de Listas

✓ Quanto à Linearidade ou Hierarquia da Estrutura

- Listas Lineares (Listas Seqüenciais, Encadeadas, Pilhas e Filas)
- Listas Não-Lineares (Árvores)

✓ Quanto à Alocação de Memória

- Estática
- Dinâmica

✓ Quanto à Natureza de Manipulação dos Elementos

- Seqüencial
- Encadeada

8 – ALOCAÇÃO ESTÁTICA DE MEMÓRIA

8.1 – Lista Estática Seqüencial

A lista estática seqüencial é uma lista linear implementada como um arranjo de registros onde estão estabelecidas regras de precedência entre os seus elementos e se configura como uma coleção ordenada de componentes do mesmo tipo. Ex: Lista de Telefones, Lista de Alunos,...

✓ Características

- Os elementos na lista estão ordenados e armazenados fisicamente em posições consecutivas;
- A inserção de um elemento na posição i causa o deslocamento a direita dos elementos de i até o último;
- A eliminação do elemento i requer o deslocamento a esquerda dos elementos $(i+1)$ até o último;
- Uma lista estática seqüencial L , ou é vazia ou pode ser descrita como:

$L =$

índice (i)	1	2	3	n
elemento	a_1	a_2	a_3	a_n

onde:

- a_1 é o primeiro elemento da lista;
- a_i precede a_{i+1} ;
- a_n é o último elemento da lista.

✓ Vantagens

- Acesso direto indexado a qualquer elemento da lista;
- Tempo constante para acessar o elemento da posição i .

✓ Desvantagens

- Movimentação de elementos nas operações de inserção e eliminação;
- O tamanho máximo da lista deve ser pré-estimado.

✓ Quando usar

- Listas pequenas;
- O tamanho máximo da lista for bem definido;
- Manipulação de dados com características de inserção e eliminação no final da lista.

✓ Implementação

TAD (Tipo Abstrato de Dados)

CONST

max = ----;

TIPO


Lista = REGISTRO

elem : VETOR [1..max] de T;

num : 0..max;

FIM;

Tipo de Dado
inteiro, real,
logico ou caracter



VAR

L : Lista;

Exemplos:

a) Rotina para criar uma lista estática seqüencial;

```
Procedimento CRIA_LISTA (var L: Lista);  
Início  
    L.num  $\leftarrow$  0;  
Fim;
```

b) Rotina para verificar se a lista está ou não vazia;

```
Funcao VAZIA (L: Lista) : logico;  
Início  
    se (L.num = 0)  
        entao VAZIA  $\leftarrow$  verdadeiro  
        senao VAZIA  $\leftarrow$  falso  
    fim se;  
Fim;
```

c) Rotina para retornar o primeiro elemento da lista.

```
Funcao PRIMEIRO (L: Lista) : T;  
Início  
    PRIMEIRO  $\leftarrow$  L.elem[1];  
Fim;
```

Exercícios

- 1) Escreva uma rotina que retorne o último elemento da lista.
- 2) Escreva uma rotina para inserir um elemento na lista.
- 3) Escreva uma rotina que retorne a quantidade de elementos da lista.
- 4) Escreva uma rotina para eliminar um elemento da lista.
- 5) Escreva uma rotina que verifique se os elementos da lista estão ordenados.

8.2 – Lista Estática Encadeada

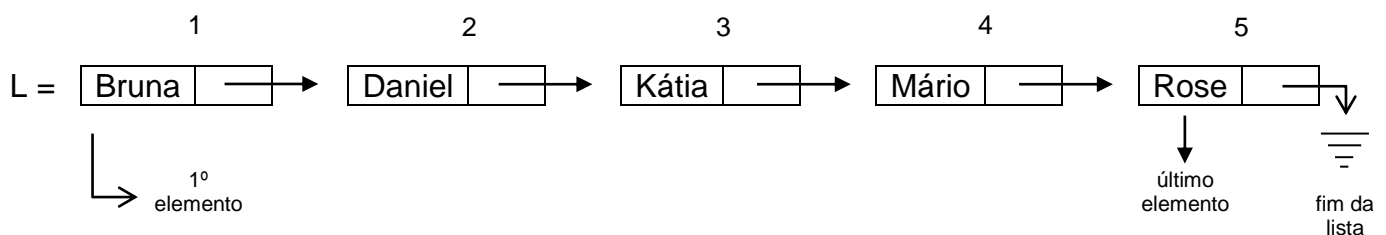
Os elementos de uma lista estática encadeada são registros com um dos componentes (campos) destinados a guardar o endereço do próximo registro, possibilitando assim o encadeamento dos elementos da lista. Cada registro tem o seguinte formato:

valor	lig
-------	-----

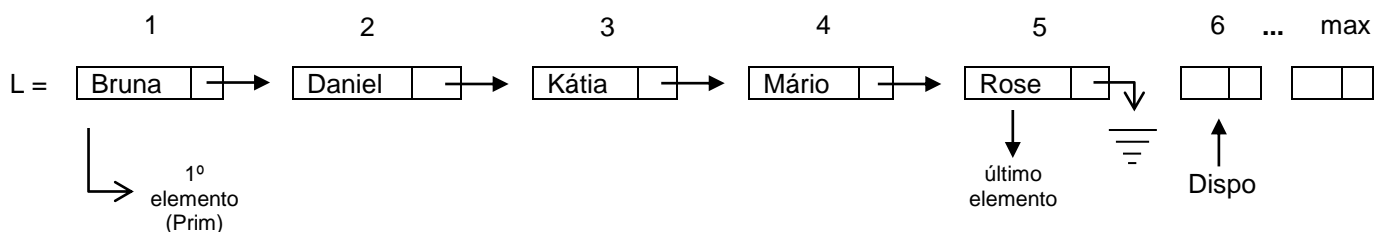
Ex:

Alessandra	2
------------	---

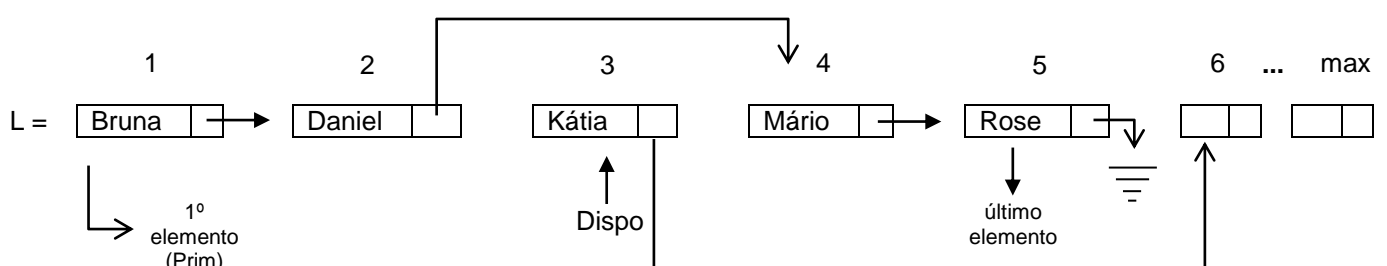
Uma lista hipotética L possui os seguintes elementos: Bruna, Daniel, Kátia, Mário e Rose. Então, pode-se representar a lista L da seguinte forma:



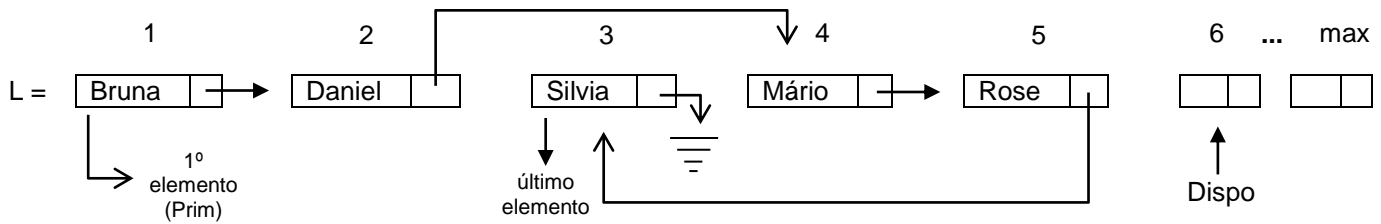
Para gerenciar esse tipo de lista deve-se trabalhar com dois conjuntos de informação: os elementos da lista (armazenados) e as posições da lista que ainda estão disponíveis. Obs: Considera-se como “fim da lista” o último elemento do vetor que tenha valor armazenado, que pode não ser necessariamente o tamanho máximo do vetor. Convencionou-se a utilizar o valor 0 (zero) para referenciar o fim da lista. Assim tem-se:



Como exemplo o elemento com o valor “Kátia” será eliminado da lista. Nesse caso, o registro desta posição ficará disponível e pronto para ser usado novamente. Assim, a referência desse registro deve passar a integrar a lista denominada por “Dispo” que contém os registros ainda livres (disponíveis) da lista.



Se numa próxima inserção houver a necessidade de inserir o valor “Silvia” na lista, esta deve ficar configurada da seguinte forma:



Deve ficar claro que o que importa é o encadeamento dos elementos e não a ordem seqüencial física do armazenamento.

✓ Vantagens

- Não requer a movimentação dos elementos nas operações de inserção e eliminação;
- Apenas os campos de ligação são manipulados e alterados.

✓ Desvantagens

- Necessário prever espaço máximo da lista;
- Necessidade de gerenciar o campo Dispo;
- O acesso não é indexado;
- Aumento no tempo de execução;
- Reserva de espaço para Dispo.

✓ Quando usar

- Quando for possível fazer uma boa previsão do espaço utilizado;
- Quando o ganho dos movimentos dos elementos sobre a perda de acesso direto a cada elemento for compensatório.

✓ Implementação

TAD (Tipo Abstrato de Dados)

CONST

max = ----;

TIPO

endereço = 0..max;

Reg = REGISTRO

info : T;

lig : endereço;

FIM;

Lista = REGISTRO

A : VETOR [1..max] de Reg;

prim, dispo : endereço;

FIM;

VAR

L : Lista;

✓ Operações sobre listas seqüenciais encadeadas

- Inicialização da lista;
- Inserção com a lista vazia e não vazia;
- Inserção antes ou após o registro x;
- Acesso ao primeiro elemento da lista;
- Acesso ao último elemento da lista;
- Verificação de lista vazia ou cheia;
- Eliminação de elementos da lista;
- Qtde de elementos da lista.

Exercícios

- 1) Escreva uma rotina para inicializar uma lista seqüencial encadeada.
- 2) Escreva uma rotina que retorne se a lista está vazia.
- 3) Escreva uma rotina que retorne se a lista está cheia.
- 4) Escreva uma rotina para retornar o primeiro elemento da lista.
- 5) Escreva uma rotina para retornar o último elemento da lista.
- 6) Escreva uma rotina para retornar a quantidade de elementos da lista.
- 7) Escreva uma rotina para obter determinado nó (posição) da lista.
- 8) Escreva uma rotina para devolver determinado nó da lista.
- 9) Escreva uma rotina para inserção de elementos em lista vazia.
- 10) Escreva uma rotina para inserção de elementos em lista não vazia.
- 11) Escreva uma rotina para inserção de elementos após o registro (nó, elemento) k.
- 12) Escreva uma rotina para eliminar um elemento da lista.

Esboço do T.A.D

L =							
	1	2	3	...	N	...	max
	A	<div>Info lig</div>	<div>Info lig</div>	<div>Info lig</div>		<div>Info lig</div>	<div>Info lig</div>
	prim						
	dispo						

9 – ALOCAÇÃO DINÂMICA DE MEMÓRIA

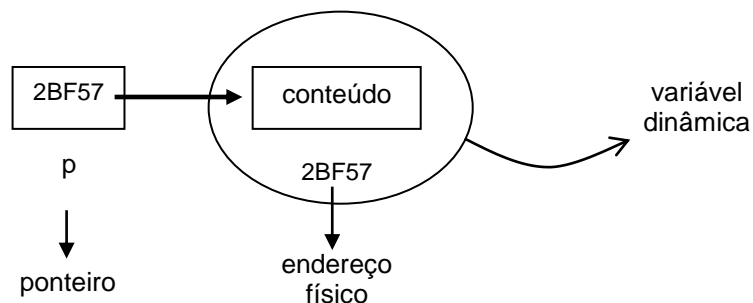
9.1 – Ponteiros

Na alocação estática de memória, como já foi abordada, há a necessidade do programador fazer uma estimativa prévia da quantidade de memória a ser ocupada pelos dados de sua aplicação. Isso se deve porque as estruturas de dados estáticas necessitam ser declaradas antes do início do código, sendo assim, chamadas de estruturas pré-referenciadas. A partir daí o programador não tem muita flexibilidade para mudar essa situação.

Em contrapartida, na alocação dinâmica de memória o programador dispõe de estruturas de dados pós-referenciadas, criando novos registros sob demanda e necessidade da aplicação, ficando limitada apenas em relação à quantidade de memória física disponível no hardware.

As linguagens de programação modernas tornaram possível explicitar não apenas os dados, mas também os endereços desses dados. Isso é possível com a utilização de ponteiros, que são variáveis de memória que “apontam” para endereços de memória. Esses endereços de memória apontados pelos ponteiros são conhecidos como variáveis dinâmicas.

Exemplo:



A notação introduzida por Wirth para a linguagem de programação Pascal é:

Type

$tp = ^T;$

Essa declaração expressa que valores do tipo tp são ponteiros para dados do tipo T . Portanto, lê-se o símbolo “ $^$ ” como “ponteiro para”.

Valores para ponteiros são gerados quando dados correspondentes a seus tipos são alocados/desalocados dinamicamente. Em pascal os procedimentos `new()` e `dispose()` existem com esse propósito.

Em pseudolinguagem serão adotados os comandos:

ALOCA: cria um ponteiro que automaticamente aponta para um endereço de memória existente. Ex:

$ALOCA(p);$ =
The diagram shows a box labeled 'p' with an arrow pointing to a box labeled 'variável dinâmica'.

DESALOCA: destrói um ponteiro juntamente com a variável dinâmica associada. Ex:

$DESALOCA(p);$ =
The diagram shows a box labeled 'p' and a box labeled 'variável dinâmica', both crossed out with a large 'X'.

Exemplo:

ALGORITMO EXEMPLO;

TIPO

p = ^ character;

VAR

nome : p;

INICIO

ALOCA (nome);

leia (nome^);

exiba (nome^);

DESALOCA (nome);

FIM.

Exercício: Usando os conceitos de ponteiro e variáveis dinâmicas, escreva um algoritmo que armazene o nome de um aluno e suas duas notas de prova. Calcule a média das notas e exiba o nome e a média.

9.2 – Listas Dinâmicas

A implementação de listas dinâmicas é feita com o uso de ponteiros e dessa forma o trabalho de alocação de memória fica facilitado. É possível então criar registros por meio de ponteiros e, conforme a necessidade e a demanda, acrescentando elementos ou eliminando posições no arranjo de dados ou lista dinâmica.

✓ Operações sobre Listas Dinâmicas

- Criar lista vazia;
- Inserir o primeiro elemento na lista;
- Inserir elemento no início da lista;
- Acessar o primeiro elemento;
- Acessar o último elemento;
- Quantidade de elementos da lista;
- Inserir valor (após determinado elemento ou na posição x);
- Eliminar elemento (após determinado elemento ou da posição x).

✓ Registros com ponteiros

Uma variável do tipo p_rec (por exemplo), aponta para ou é um ponteiro para um registro do tipo de dados registro, conforme trecho de declaração da estrutura de dados abaixo:

TAD (Tipo Abstrato de Dados)

TIPO

p_rec = ^rec;

Lista = p_rec;

rec = REGISTRO

info : T;

lig : Lista;

Fim;

VAR

P, L : Lista; {ponteiros}

Obs: O tipo de dados ponteiro é o único tipo que pré-referencia outro tipo em Pascal.

Um ponteiro do tipo “Lista” (definido na estrutura acima) pode assumir o conjunto de valores que corresponde a endereços reais de memória para os campos “info” e “lig”. Exemplo:



Onde o conteúdo de P corresponde ao endereço do objeto (conjunto de campos info e lig). Esses endereços são as ligações das listas encadeadas dinâmicas.

✓ Referências – Sintaxe

- O acesso ao registro depende do tipo de alocação adotada. Se utilizar alocação em vetor, referencia-se como: L.A[p] ; e se a alocação for dinâmica, referencia-se como: p^.

- Para referenciar ponteiro, objeto e campo, as sintaxes são:

- ponteiro: p

- objeto: p^

- campos: p^.info
p^.lig

- Para referenciar e controlar a informação sobre fim da lista ou lista vazia, também se deve utilizar o valor 0 (zero), também conhecido como endereço nulo ou terra ($\overline{\equiv}$) para esse fim.

Obs: A linguagem de programação Pascal possui uma constante pré-definida para denotar esse endereço nulo chamada “nil”.

✓ Ponteiro X Objeto apontado

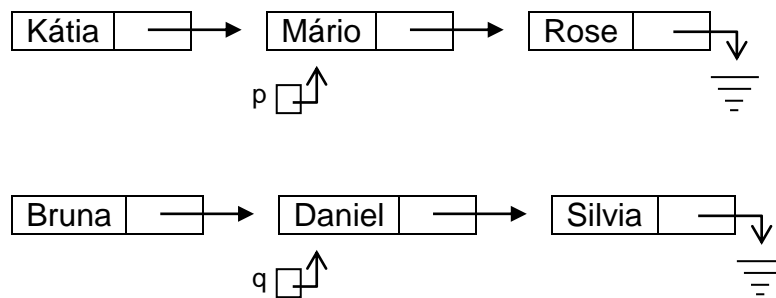
Ex:

a) $p \leftarrow q$; (*ponteiros*)

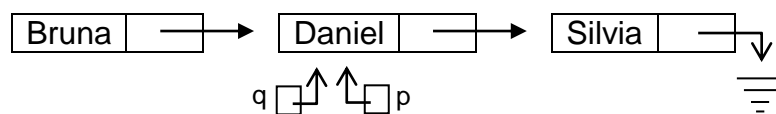
b) $p^{\wedge} \leftarrow q^{\wedge}$; (*conteúdo dos registros apontados pelos ponteiros*)

Ex:

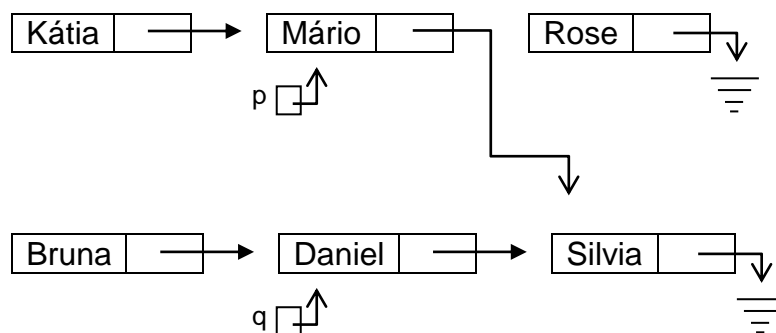
Situação Inicial



a) $p \leftarrow q$;



b) $p^{\wedge} \leftarrow q^{\wedge}$;



✓ Manipulação de Registros

1) Criação de um registro

Substitui o procedimento Obter_No (j), utilizado em listas estáticas encadeadas, dada uma variável do tipo ponteiro para tipo $\wedge\text{rec}$.



- Efetivamente aloca uma variável do tipo rec;
- Gera um ponteiro $\wedge\text{rec}$ apontado para aquela variável;
- Atribui o ponteiro à variável p.

A partir daí:

- O ponteiro pode ser referenciado como p;
- A variável referenciada por p é denotada por p^\wedge .

2) Liberação de um registro

Substitui o procedimento Devolver_No (j).



- Esta operação libera o espaço apontado por p;
- p passa a ter valor indefinido.

Exercícios

- Escreva uma rotina para criar uma lista.
- Escreva uma rotina para inserir o primeiro elemento.
- Escreva uma rotina para inserir um elemento no início da lista.
- Escreva uma rotina que retorne o primeiro elemento da lista.
- Escreva uma rotina que retorne o último elemento da lista.
- Escreva uma rotina para retornar a quantidade de elementos da lista.
- Escreva as rotinas para a) inserir valor v depois do elemento x; e b) Inserir valor v na posição x.
- Escreva uma rotina para eliminar o primeiro elemento da lista.
- Escreva as rotinas para a) eliminar o elemento após x; e b) eliminar o elemento da posição x.

Exercícios

1) Diferencie Alocação Estática de Memória de Alocação Dinâmica de Memória.

2) Explique o que são ponteiros.

3) O que são listas? Explique e dê exemplos.

4) Defina, sintaticamente:

a) ponteiro

b) objeto

c) campo

5) Explique os seguintes comandos de atribuição:

a) $p \leftarrow q$;

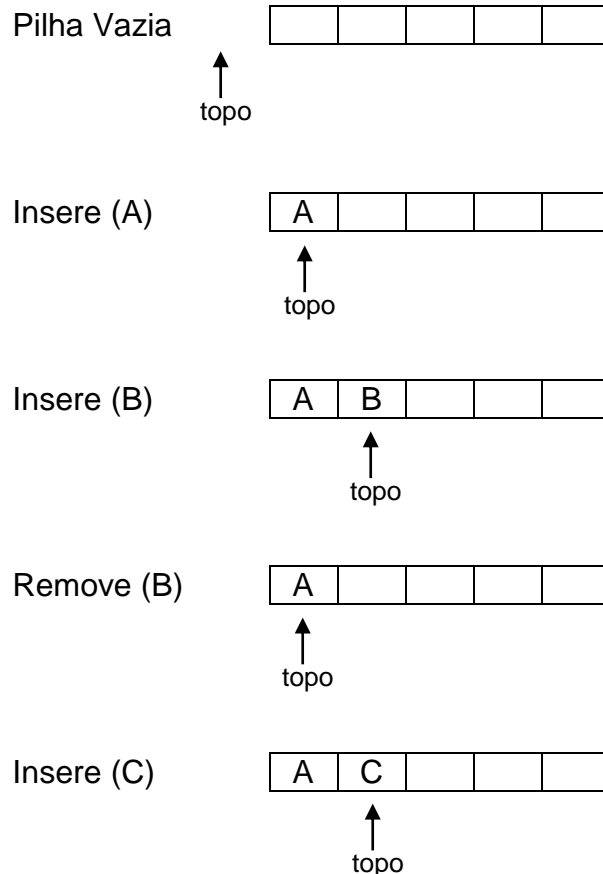
b) $p \leftarrow p.\text{lig}$;

c) $p \leftarrow q$;

10 – PILHAS

Pilhas são listas onde as inserções de novos elementos ou as eliminações de elementos já armazenados se dão em uma única extremidade da lista: no topo.

Pilhas também são conhecidas como Listas LIFO (Last In First Out – Último que Entra, Primeiro que Sai), ou seja, o último elemento inserido é o primeiro a ser acessado, lido, excluído, processado,...

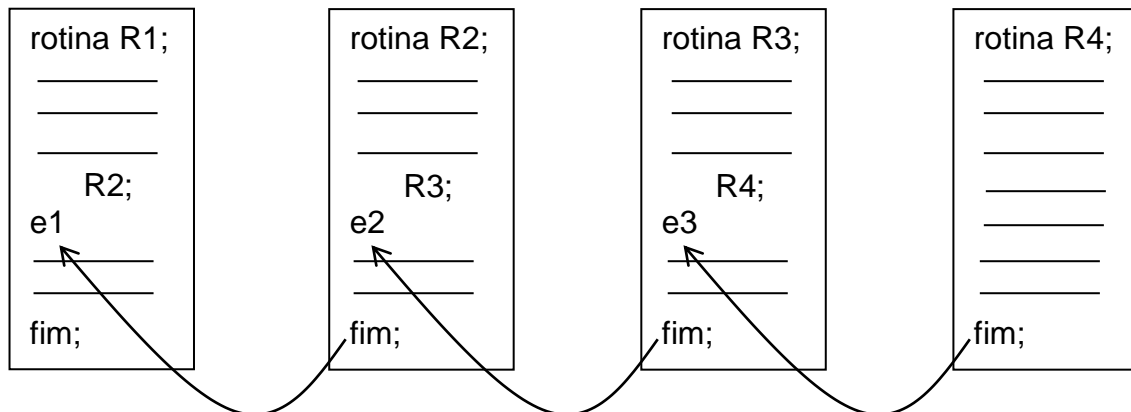


✓ Operações sobre Pilhas

- Criar (P) – criar uma pilha vazia;
- Inserir (x,P) – insere x no topo da pilha P (empilha) ou (push);
- Vazia (P) – testa se a pilha está vazia;
- Topo (P) – acessa o elemento do topo da pilha;
- Remove (P) – elimina o elemento do topo da pilha P (desempilha) ou (pop).

✓ Aplicações de Pilhas

Exemplo: Chamadas de Rotinas



Quando a rotina R1 é executada ela efetua uma chamada a rotina R2, que deve carregar consigo o endereço de retorno e1. Ao término de R2 o processamento deve retornar a rotina R1 no devido endereço. Situação idêntica ocorre em R2 e R3.

Assim, quando um procedimento termina, é o seu endereço de retorno que deve ser consultado. Portanto, há uma lista implícita de endereços (e0, e1, e2, e3) que deve ser manipulada como uma pilha pelo sistema operacional para controle na execução de programas. *Obs: e0 é o endereço de retorno de R1.*

✓ Implementação de Pilhas

Como implementar uma pilha? Como lista seqüencial ou encadeada? No caso geral de listas ordenadas, a maior vantagem da alocação encadeada sobre a seqüencial (se a memória não for problema) é a eliminação dos deslocamentos necessários quando da inserção ou eliminação dos elementos.

No caso de pilhas, essas operações de deslocamentos não ocorrem. Portanto, pode-se afirmar que a alocação seqüencial é mais vantajosa na maioria das vezes.

✓ Alocação Seqüencial de Pilhas

TAD (Tipo Abstrato de Dados)

CONST

max_pilha = ----;

TIPO

indice = 0..max_pilha;

Pilha = VETOR [1..max_pilha] de T;

VAR

P : Pilha;

topo : indice;

Exercícios

- 1) Escreva uma rotina para criar uma pilha.
- 2) Escreva uma rotina para verificar se a pilha está vazia.
- 3) Escreva uma rotina para verificar se a pilha está cheia.
- 4) Escreva uma rotina para inserir um elemento na pilha.
- 5) Escreva uma rotina para eliminar um elemento da pilha.
- 6) Escreva uma rotina para acessar um elemento da pilha.
- 7) Escreva uma rotina para acessar um elemento da pilha e eliminá-lo em seguida.

✓ Alocação Encadeada de Pilhas

TAD (Tipo Abstrato de Dados)

TIPO

pont = ^reg_pilha;

Pilha = pont;

reg_pilha = REGISTRO

info : T;

lig : pont;

FIM;

VAR

P : Pilha;

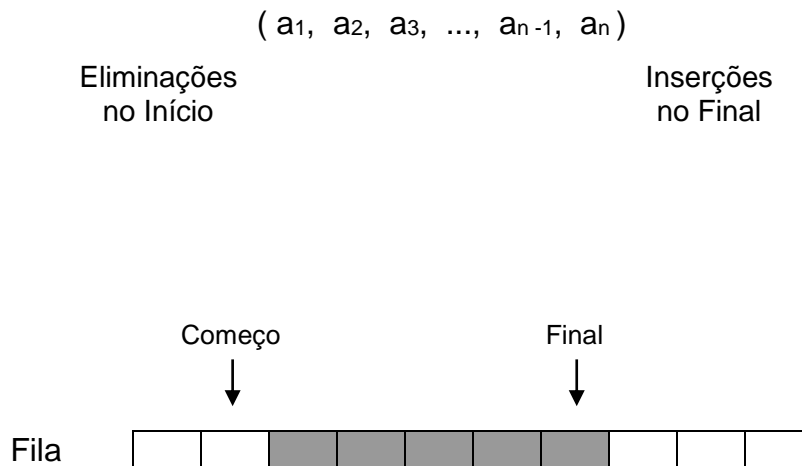
Exercícios

- 1) Escreva uma rotina para criar uma pilha.
- 2) Escreva uma rotina para verificar se a pilha está vazia.
- 3) Escreva uma rotina para inserir um elemento na pilha.
- 4) Escreva uma rotina para eliminar um elemento da pilha.
- 5) Escreva uma rotina para acessar um elemento da pilha.
- 6) Escreva uma rotina para acessar um elemento da pilha e eliminá-lo em seguida.

11 – FILAS

Filas são listas em que as inserções de novos elementos são feitas numa extremidade da lista e as eliminações na outra.

Filas também são conhecidas como Listas FIFO (First In First Out – Primeiro que Entra, Primeiro que Sai), ou seja, o primeiro elemento inserido é o primeiro a ser acessado, lido, excluído, processado,...



✓ Operações sobre Filas

- Criar (F) – cria uma fila vazia;
- Inserir (x,F) – insere x no final da fila;
- Vazia (F) – testa se a fila esta vazia;
- Primeiro (F) – acessa o elemento do inicio da fila;
- Ultimo (F) – acessa o elemento do final da fila;
- Elimina (F) – elimina o elemento do inicio da fila.

✓ Aplicações de Filas

Exemplo: Escalonamento de jobs (fila de processos aguardando recursos do S.O.)

✓ Implementação de Filas

Como implementar uma fila? Como lista seqüencial ou encadeada? Estática ou Dinâmica? Só tem sentido se falar em fila seqüencial estática ou fila encadeada dinâmica, uma vez que não existe movimentação de elementos. As filas encadeadas são usadas quando não há previsão do tamanho máximo da fila.

✓ Alocação Estática Seqüencial de Filas

TAD (Tipo Abstrato de Dados)

CONST

max_fila = ----;

TIPO

indice = 0..max_fila;

Fila = VETOR [1..max_fila] de T;

VAR

F : Fila;

Começo, {posição anterior ao primeiro elemento}

Final : indice; {posição do último elemento}

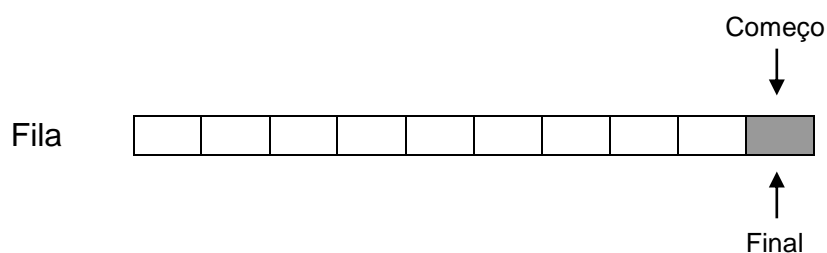
Exercícios

- 1) Escreva uma rotina para criar uma fila.
- 2) Escreva uma rotina para verificar se a fila está vazia.
- 3) Escreva uma rotina para verificar se a fila está cheia.
- 4) Escreva uma rotina para acessar o primeiro elemento na fila.
- 5) Escreva uma rotina para acessar o último elemento da fila.
- 6) Escreva uma rotina para inserir um elemento na fila.
- 7) Escreva uma rotina para eliminar um elemento da fila.

✓ Problemas na Alocação Estática Seqüencial de Filas

O que aconteceria com a fila considerando a seguinte seqüência de operações? I, E, I, E, I, E, I, E. Legenda: I – Inclusão; E – Exclusão.

Resp. → A fila terá sempre 1 elemento ou nenhum, e no entanto, num certo momento:



Ou seja, apenas um elemento (ou nenhum) na fila, ocupando a última posição do vetor. Na próxima inserção ocorrerá uma condição de overflow e na verdade a fila está vazia!

Exercício: Implemente uma solução para resolver o problema.

Solução: Na rotina de eliminação, após a atualização da variável Começo, verificar se a fila ficou vazia, isto é, se Começo = Final. Neste caso, reinicializar: Começo e Final \leftarrow 0.

Procedimento ELIMINA (var Comeco, Final: indice);

Inicio

se (VAZIA = falso)

entao Comeco \leftarrow Comeco + 1;

se (Comeco = Final)

entao Comeco \leftarrow 0;

Final \leftarrow 0;

fim se;

fim se;

Fim;

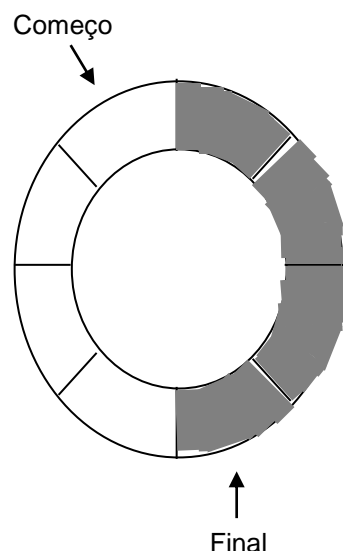
E o que aconteceria com a fila se a seqüência de operações fosse: I, I, E, I, E, I, E, I, E, I?

Resp. \rightarrow A fila estaria com no máximo dois elementos e ainda assim ocorreria overflow com a fila quase vazia.

Alternativa: Forçar Final a usar o espaço liberado por Comeco, implementando o que se conhece por Fila Circular!

✓ Fila Circular

Fila Circular implementada como um Anel.



Condições:

- Índices do vetor: $0..max_fila - 1$;
- Ponteiros de controle: *comeco* e *final*;
- Inicialmente: *comeco* = *final* = 0;
- Quando *Final* = $max_fila - 1$, o próximo elemento será inserido na posição 0 (se essa posição estiver vazia!);
- Condição de fila vazia: *Comeco* = *Final*.

T.A.D.

TIPO

indice = $0..max_fila - 1$;
Fila = VETOR [$0..max_fila - 1$] de T;

VAR

F: Fila;
Comeco, *Final* : *indice*;

Procedimento INSERE (var F: Fila; var *Final*: *indice*; valor: T);

Inicio

Final \leftarrow (*Final* + 1) MOD *max_fila*;
F [*Final*] \leftarrow valor;

Fim;

Procedimento ELIMINA (var *Comeco*: *indice*);

Inicio

Comeco \leftarrow (*Comeco* + 1) MOD *max_fila*;

Fim;

Procedimento INSERE (var F: Fila; var *Final*: *indice*; *Comeco*: *indice*; valor: T);

Inicio

se ((*Final* + 1) MOD *max_fila* = *Comeco*)
 entao exiba ("Fila Cheia!")
 senao
 Final \leftarrow (*Final* + 1) MOD *max_fila*;
 F [*Final*] \leftarrow valor;

fim se;

Fim;

✓ Alocação Dinâmica Encadeada de Filas

TAD (Tipo Abstrato de Dados)

TIPO

```
pont = ^reg_fila;  
Fila = pont;  
reg_fila = REGISTRO  
    info : T;  
    lig : pont;  
    FIM;
```

VAR

```
Comeco, Final : Fila;
```

Exercícios

- 1) Escreva uma rotina para criar uma fila.
- 2) Escreva uma rotina para verificar se a fila está vazia.
- 3) Escreva uma rotina para inserir o primeiro elemento na fila.
- 4) Escreva uma rotina para inserir um elemento na fila.
- 5) Escreva uma rotina para acessar o primeiro elemento da fila.
- 6) Escreva uma rotina para acessar o último elemento da fila.
- 7) Escreva uma rotina para eliminar um elemento da fila.