UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CIÊNCIAS EXATAS E TECNOLÓGICAS – Curso: Informática / Ciência da Computação

Programação II

Disciplina: Linguagem de Programação PASCAL **E-mail:** ***osorio@exatas.unisinos.br***
Professor responsável: ***Fernando Santos Osório*** **Web:**
Semestre: 2004/2 ***http://www.inf.unisinos.br/~osorio/prog2.html***
Horário: 63 **Xerox : *Pasta 54* – LAB. II (Xerox do "Alemão")**

Lista Linear Seqüencial com Alocação Estática

1. Alocação de Memória e Estruturas de Dados:

Ao criar um programa em Pascal usualmente temos que especificar, antes de começar a executar o programa, as variáveis que vamos usar, reservando assim um espaço na memória. As variáveis que são alocadas em posições fixas da memória são chamadas de *variáveis estáticas*, e as variáveis que não possuem uma posição fixa, e que são criadas e destruídas durante a execução do programa, são chamadas de *variáveis dinâmicas*.

A alocação de memória no computador pode ser dividida em dois grupos principais:

- Alocação Estática: os dados tem um tamanho fixo e estão organizados seqüencialmente na memória do computador. Um exemplo típico de alocação estática são as variáveis globais, e os vetores (*arrays*).
- Alocação Dinâmica: os dados não precisam ter um tamanho fixo, pois podemos definir para cada dado quanto de memória que desejamos usar. Sendo assim vamos alocar espaços de memória (blocos) que não precisam estar necessariamente organizados de maneira seqüencial, podendo estar distribuídos de forma esparsa na memória do computador. Na alocação dinâmica, vamos pedir para alocar/desalocar blocos de memória, de acordo com a nossa necessidade, reservando ou liberando blocos de memória durante a execução de um programa. Para poder "achar" os blocos esparsos na memória usamos as variáveis do tipo *Ponteiro* (indicadores de endereços de memória).

Vamos estudar aqui a construção de estruturas de dados em Pascal, usando estes dois tipos de métodos de alocação de memória: estática e dinâmica. O primeiro tipo de estrutura de dados a ser abordado, que é muito usado em Pascal, são as listas. As listas servem para armazenar um conjunto de dados, sejam eles agrupados (vetores = *arrays*) ou não (listas encadeadas).

1.1. Conjunto de dados com alocação estática:

A) *Listas lineares seqüenciais – Listas usando Vetores Simples*
B) *Pilhas – LIFO (Last In / First Out)*
C) *Filas – FIFO (First In / First Out)* [Fila Circular]
D) *Deques – Double Endend Queue*

1.2. Conjuntos de dados com <u>alocação dinâmica</u>:

A) *Listas simplesmente encadeadas*
B) *Listas duplamente encadeadas*
C) *Filas*
D) *Pilhas*
E) *Deques*
F) *Árvores:*
Árvore binária ordenada / não ordenada
Árvore binária ordenada balanceada / não balanceada
Árvore ternária, quaternária, ..., Genérica)

2. Alocação Estática de Memória:

A alocação estática tem a vantagem de manter os dados organizados na memória, dispostos um ao lado dos outros de forma linear e seqüencial. Isto facilita a sua localização e manipulação, em contrapartida, precisamos estabelecer previamente a quantidade máxima necessária de memória para armazenar uma determinada estrutura de dados. Por exemplo, se quisermos construir um cadastro usando uma lista linear seqüencial, precisamos primeiramente definir o seu tamanho máximo (tamanho do array), onde este tamanho só poderá ser alterado se editarmos o fonte do programa e recompilarmos este.

Vetor: Array [1..10] of Dado;

Dado 01	Dado 02	Dado 03	Dado 04	Dado 05	Dado 06	•••	Dado 10
[1]	[2]	[3]	[4]	[5]	[6]		[10]

Imagine se um editor de texto, como o editor Word da Microsoft, tivesse que "prever" com antecedência a quantidade de linhas do arquivo que um usuário pretende editar, sem reservar memória demais pois iria bloquear desnecessariamente uma grande quantidade da memória da máquina (que outras aplicações não poderiam usar), e sem definir um espaço muito pequeno pois não seria adequado parar a edição do texto por falta de memória reservada para armazenar os dados (mesmo tendo ainda espaço disponível na memória do micro). E se fosse necessário alterar a quantidade de memória prevista, seria necessário editar o código fonte e recompilar o programa novamente...

Resumindo:
Vantagem da alocação estática: Simplicidade de uso e programação
Desvantagem da alocação estática: Limitação prévia dos recursos alocados

As estruturas de dados usadas para armazenar um conjunto de dados (exemplo: um cadastro com vários registros), podem ser organizadas de formas diferentes, de acordo com **a maneira que os dados são <u>inseridos</u> e <u>retirados</u> da memória do computador**. A forma mais simples é implementada através das listas lineares seqüenciais (vetores simples). Modos específicos de inserção e remoção dos dados definem as estruturas de dados, que são largamente usadas na computação, as chamadas: pilhas, filas e deques.

Vamos a seguir definir ***rotinas genéricas*** de manipulação destas estruturas de dados.

Listas lineares seqüenciais

As listas lineares seqüenciais são vetores, onde a **inserção** usualmente pode ser realizada de uma das seguintes maneiras:

1) Inserção no final da lista de valores previamente inseridos. Exemplo:

Lista Linear Seqüencial => Vetor: Array [1..10] of Integer; // Inicio =1, Fim = 0 (Fim=Qtde.)

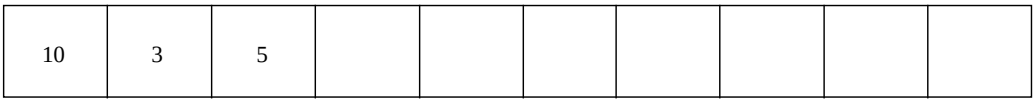
Insere_lista (Vetor, Fim, 10);

[1] = Início, Fim

Insere_lista (Vetor, Fim, 3);

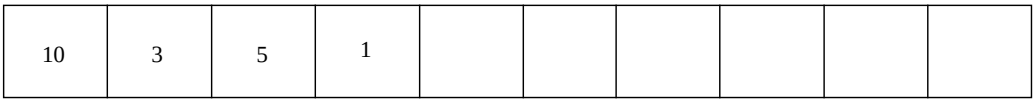
[1] = Início [2] = Fim

Insere_lista (Vetor, Fim, 5);



[1] = Início [3] = Fim

Insere_lista (Vetor, Fim, 1);



[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
[1] = Início [4] = Fim

Neste modo de inserção é necessário controlar o total de elementos inseridos (Total, ou Quantidade ou Fim), onde cada novo dado é sempre inserido após o último dado inserido, desde que o vetor ainda possua posições livres. Rotina de inserção em um vetor (LLS):

```
Const    Tamanho_Maximo = 10;
Type     Vetor = Array [1..Tamanho_Maximo] of Integer;

Function Insere_lista (var V: Vetor; var Qtde: Integer; Dado: Integer): boolean;
Begin
   IF  Qtde = Tamanho_Maximo
   THEN  Insere_lista := False
   ELSE  Begin
              V[Qtde] := Dado;
              Qtde := Qtde + 1;
              Insere_lista:=True;
         End;
End;
```

2) Inserção ordenada dos dados. Exemplo:

Lista Linear Seqüencial => Vetor: Array [1..10] of Integer; // Inicio =1, Fim = 0 (Fim=Qtde.)

Insere__ord_lista (Vetor, Fim, 10);

10									

[1] = Início, Fim

Insere_ord_lista (Vetor, Fim, 3);

3	10								

[1] = Início [2] = Fim

Insere_ord_lista (Vetor, Fim, 5);

3	5	10							

[1] = Início [3] = Fim

Insere_ord_lista (Vetor, Fim, 1);

1	3	5	10						
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

[1] = Início [4] = Fim

Neste modo de inserção além de ser necessário controlar o total de elementos inseridos (Total, ou Quantidade ou Fim), cada novo dado é sempre inserido de forma ordenada, o que implica em "abrir espaço" para que um dado seja inserido na posição correta. Sempre testando se o vetor ainda possui posições livres.

EXERCÍCIO Nro. 1:

1) Crie uma rotina de inserção ordena em vetores, de acordo com o exemplo acima. Esta rotina deve inserir dados numéricos inteiros de forma ordenada crescente.

EXERCÍCIO Nro. 2:

2) Faça um programa que leia um total de 15 números reais e armazene em um vetor (LLS – Lista Linear Seqüencial), salvando posteriormente em disco o vetor criado em um arquivo texto. Procure fazer este programa da forma mais genérica e modular possível, de modo que as rotinas possam ser reaproveitadas posteriormente em outros programas.

Listas lineares seqüenciais (Continuação...)

As listas lineares seqüenciais são vetores, onde a **remoção** usualmente pode ser realizada de uma das seguintes maneiras:

1) Remoção de um dado, movendo o último valor da lista de valores para a posição do valor que foi removido ("tapar o buraco" do vetor). Este tipo de remoção não preserva o ordem original da seqüência dos dados. Exemplo:

Lista Linear Seqüencial => Vetor: Array [1..10] of Integer;

Remove_lista (Vetor, Fim, 2); { Remove o segundo elemento da lista }

8	**4**	1	10	3	5	***12***			

[1] = Início [7] = Fim

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
8	***12***	1	10	3	5				

[1] = Início [6] = Fim

```
Const    Tamanho_Maximo = 10;
Type     Vetor = Array [1..Tamanho_Maximo] of Integer;

Function Remove_lista (var V: Vetor; var Qtde: Integer; Qual: Integer): boolean;
Begin
   IF  Qual > Qtde
   THEN  Remove_lista := False
   ELSE  IF  Qual = Qtde
            THEN  Begin
                     Qtde:=Qtde – 1;
                      Remove_lista := True;
                  End
            ELSE   Begin
                      V[Qual] := V[Qtde];
                      Qtde := Qtde - 1;
                      Remove_lista := True;
                   End;
End;
```

2) Remoção de um dado, deslocando ("puxando" para o início) todos os dados seguintes ao dado que foi removido. Este tipo de remoção preserva a ordem original dos dados, mas entretanto é mais custoso do ponto de vista computacional, pois se o vetor possuir muitos dados teremos que realizar um grande número de deslocamentos. Exemplo:

Remove_lista (Vetor, Fim, 2); { Remove o segundo elemento da lista }

1	**3**	5	6	8	9	***12***			

[1] = Início [7] = Fim

1	5	6	8	9	12				

[1] = Início [6] = Fim

3) Remoção lógica de um dado. Neste tipo de remoção, o dado não é verdadeiramente eliminado do vetor, apenas é colocada uma marca de exclusão lógica (ex. flag), indicando que este dado deve ser desprezado em outras operações (busca, exibição, etc). A exclusão lógica tem o inconveniente de não liberar o espaço de memória que o dado estava ocupando, mas entretanto permite que o dado removido seja recuperado ("undelete") e inclusive podendo em um momento posterior os diversos dados marcados como removidos ser eliminados todos de uma vez ("pack" – compacta os "buracos" existentes no vetor, eliminando fisicamente os dados). Exemplo:

Lista Linear Seqüencial:
Vetor => Dado: Array [1..10] of Integer;
Removido: Array [1..10] of Boolean;

Remove_lista (Vetor, Fim, 2); { Remove o segundo elemento da lista }

8	**4**	1	10	3	5	12				Dado
False	False	False	False	False	False	False				Removido

[1] = Início [7] = Fim

8	**4**	1	10	3	5	12				Dado
False	**True**	False	False	False	False	False				Removido
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	

[1] = Início [7] = Fim

Note que após a remoção a quantidade total de elementos do vetor (fim) não foi alterada.

EXERCÍCIO Nro. 3:

3) Crie uma rotina de remoção que preserve a ordenação dos dados, realizando o deslocamento dos dados, conforme descrito no item 2 (remoção em vetores).

EXERCÍCIO Nro. 4:

4) Crie uma rotina de remoção em vetores, de acordo com o exemplo acima que descreve a remoção lógica (item 3). Esta rotina deve realizar a exclusão lógica de dados numéricos inteiros contidos no vetor.

EXERCÍCIO Nro. 5:

5) Faça um programa que leia o nome e a média final de um total de 15 alunos e armazene em um vetor (LLS – Lista Linear Seqüencial), salvando posteriormente em disco este vetor que foi criado em um arquivo texto denominado "alunos.txt". Solicite para o usuário que ele indique o valor mínimo de aprovação. Considerando a nota limite para aprovação, insira os alunos reprovados em uma nova lista de reprovados, eliminando estes da lista geral de alunos. Portanto, ficaremos com 2 listas, uma contendo os aprovados (onde foram excluídos os reprovados) e uma contendo os reprovados. Salve em disco o vetor que foi criado com os alunos reprovados em um arquivo texto denominado "reprov.txt", e o vetor que contém os alunos aprovados, salve em um arquivo texto denominado "aprov.txt".
Procure fazer este programa da forma mais genérica e modular possível, de modo que as rotinas possam ser reaproveitadas posteriormente em outros programas.

Vide rotinas sugeridas no material da Aula 04 de Lab2 => http://inf.unisinos.br/~osorio/lab2.html