

Sistema Aberto de Educação



Guia de Estudo

Estrutura de Dados

1ª Parte



Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas



SABE – Sistema Aberto de Educação

**Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223**

Instituição Credenciada pelo MEC – Portaria 4.385/05

**Centro Universitário do Sul de Minas - UNIS/MG
Unidade de Gestão da Educação a Distância – GEaD**

**Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG**

Varginha/MG

Todos os direitos desta edição reservados ao Sistema Aberto de Educação – SABE.
É proibida a duplicação ou reprodução deste volume, ou parte do mesmo, sob
qualquer meio, sem autorização expressa do SABE.

001.642

G216g GARCIA, Denise Ferreira.

Guia de Estudo – Estrutura de Dados.
Denise Ferreira Garcia. Varginha: GEaD-
UNIS/MG, 2008.

47p.

1. Dados. 2. Listas Lineares. I. Título.

Atualizado e revisado por SILVA, Lázaro Eduardon em
janeiro de 2009

REITOR

Prof. Ms. Stefano Barra Gazzola

GESTOR

Prof. Ms. Tomás Dias Sant' Ana

Supervisor Técnico

Prof. Ms. Wanderson Gomes de Souza

Coord. do Núcleo de Recursos Tecnológicos

Prof^a. Simone de Paula Teodoro Moreira

Coord. do Núcleo de Desenvolvimento Pedagógico

Prof^a. Vera Lúcia Oliveira Pereira

Revisão ortográfica / gramatical

Prof^a. Silvana Prado

Design/diagramação

Prof. César dos Santos Pereira

Equipe de Tecnologia Educacional

Prof^a. Débora Cristina Francisco Barbosa

Jacqueline Aparecida da Silva













Prof. Lázaro Eduardo da Silva

Autora

DENISE FERREIRA GARCIA

Bacharel em Ciência da Computação (1997) – UNIFENAS, Alfenas - MG. Mestre em Ciência da Computação (2001) pela Universidade de Minas Gerais - UFMG, Belo Horizonte - MG. Professora desde 2000 foi também coordenadora do curso de Ciência da Computação do Centro Universitário de Formiga durante cinco anos. Atualmente é professora no mesmo curso e no curso de Engenharia de Produção do UNIFOR-MG. Atua também como professora na PUC Minas campus Arcos no curso de Sistemas de Informação.

TABELA DE ÍCONES

	REALIZE. Determina a existência de atividade a ser realizada. Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.
	PESQUISE. Indica a exigência de pesquisa a ser realizada na busca por mais informação.
	PENSE. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.
	CONCLUSÃO. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.
	IMPORTANTE. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.
	HIPERLINK. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.
	EXEMPLO. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.
	SUGESTÃO DE LEITURA. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.
	APLICAÇÃO PROFISSIONAL. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.
	CHECKLIST ou PROCEDIMENTO. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.
	SAIBA MAIS. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.
	REVENDO. Indica a necessidade de rever conceitos estudados anteriormente.

Índice de Figuras

Figura 1 - Estrutura de implementação da Pilha seqüencial.....	14
Figura 2 - Estrutura de implementação da Fila seqüencial	17
Figura 3 - Funcionamento da Fila	17
Figura 4 - Dois momentos distintos de elementos alocados em uma Fila Circular...	18
Figura 5 - Estrutura de lista linear por contigüidade dos nós.....	21
Figura 6 - Inserção de um nó na lista	22
Figura 7 - Estrutura da lista linear encadeada usando vetor.....	26
Figura 8 - Processo de inserção de um nó na lista.....	27
Figura 9 - Processo de remoção de um nó da lista	27
Figura 10 - Explicação do exemplo de alocação dinâmica de memória	35
Figura 11 - Estrutura de fila encadeada por alocação dinâmica de memória	40

Índice de Programas

Programas 1 - Declaração do tipo Pilha	14
Programas 2 - Procedimento iniciapilha	14
Programas 3 - Procedimento empilha.....	15
Programas 4 - Procedimento desempilha.....	15
Programas 5 - Declaração do tipo Fila.....	18
Programas 6 - Procedimento iniciafila	19
Programas 7 - Procedimento inserefila.....	19
Programas 8 - Procedimento retirafila	20
Programas 9 - Declaração do tipo Lista.....	22
Programas 10 - Procedimento inicialista.....	23
Programas 11 - Procedimento inserelista.....	23
Programas 12 - Procedimento removelista.....	25
Programas 13 - Declaração do tipo Lista encadeada	27
Programas 14 - Procedimento inicialista.....	28

Programas 15 - Procedimento inserelista	29
Programas 16 - Procedimento removelista.....	30
Programas 17 - Declaração do tipo apontno e no	37
Programas 18 - Procedimento iniciapilha	38
Programas 19 - Procedimento empilha.....	38
Programas 20 - Procedimento desempilha.....	39
Programas 21 - Declaração do tipo fila encadeada	40
Programas 22 - Procedimento iniciafila	41
Programas 23 - Procedimento inserefila.....	41
Programas 24 - Procedimento retirafila	42
Programas 25 - Declaração do tipo lista por alocação dinâmica de memória	43
Programas 26 - Procedimento inicialista.....	43
Programas 27 - Procedimento inserelista	44
Programas 28 - Procedimento removelista.....	45

SUMÁRIO

INTRODUÇÃO	8
DESENVOLVIMENTO: UNIDADE I	10
1.1. Tipos de Dados	10
1.2. Tipo Abstrato de Dados – TAD.....	11
1.3. Procedimentos e Funções.....	12
1.4. Pilha Seqüencial.....	13
1.5. Fila seqüencial.....	16
1.6. Lista Linear seqüencial.....	21
1.6.1. Lista Linear seqüencial por contigüidade dos nós.....	21
1.6.2. Lista Linear encadeada usando vetor	26
DESENVOLVIMENTO: UNIDADE II	32
2.1. Tipo Apontador	32
2.2. Ponteiros	32
2.2.1. Declaração	32
2.2.2. Inicialização.....	33
2.2.3. Criação de uma variável dinâmica	33
2.2.4. Destruição de uma variável dinâmica.....	33
2.2.5. Referência a uma variável dinâmica	34
2.3. Pilha usando alocação dinâmica de memória	35
2.4. Fila usando alocação dinâmica de memória	39
2.5. Lista usando alocação dinâmica de memória.....	43
REFERÊNCIAS	47

INTRODUÇÃO

Vamos iniciar nosso estudo de ED (Estrutura de Dados) observando a definição da palavra estrutura retirada do dicionário da Wikipédia.

es.tru.tu.ra, Substantivo, *feminino*

1. [algo constituído](#) de [partes distintas](#)
2. [partes](#) de um [corpo responsáveis](#) pela sua [resistência](#) e/ou [sustentação](#)
3. o [modo](#) como [partes organizadas](#) se [relacionam](#)
 - A **estrutura** da frase.
 - A **estrutura** da sociedade ainda é um mistério.
4. (*Biologia*) [modo](#) de [organização](#), [construção](#) e [arranjo](#) dos [tecidos](#) ou [órgãos](#)
5. (*Geologia*) a [composição](#) de uma [rocha](#), em [oposição](#) à sua [textura](#)
6. (*Química*) a [maneira](#) pelo qual os [átomos](#) se [organizam](#) em uma [molécula](#)
7. (*Sociologia*) o [sistema](#) de [relações](#) entre os [grupos constituintes](#) de uma [sociedade](#)
8. (*Linguística*) o [padrão](#) de [organização](#) de uma [língua](#) como um todo ou os [relacionamentos](#) entre seus [elementos](#) constituintes (como [morfemas](#), [fonemas](#), etc)
9. (*Lógica*) lista de atribuições, contendo geralmente um [universo de discurso](#), para os diversos sinais que constituem o alfabeto de um [sistema lógico](#), de modo que sejam associados a todas as [fórmulas](#) deste sistema [valores veritativos](#); modelo; interpretação

Wikipédia, a enciclopédia livre.

Em qualquer área de atuação relatada, a definição da palavra estrutura está relacionada com partes organizadas de alguma coisa e na nossa área de Sistemas de Informação não é diferente. Na computação existe uma grande necessidade de organizar e correlacionar os dados que são manipulados pelos algoritmos, para isso utiliza-se Estrutura de Dados. Dados estes que devem estar organizados (dispostos) de forma coerente, caracterizando uma forma, uma estrutura de dados.

O estudo das estruturas de dados, componente fundamental no aprendizado da computação, é a base sobre a qual muitos outros campos da ciência da computação

são construídos. Algum conhecimento das estruturas de dados é um imperativo para os estudantes que desejam trabalhar em implementação de qualquer sistema de software (DROZDEK, 2002).

Estruturas de dados e algoritmos estão intimamente ligados. Não se pode estudar estrutura de dados sem considerar os algoritmos associados a elas, assim como a escolha dos algoritmos em geral depende da representação e da estrutura dos dados (ZIVIANE, 2005).

Sabemos que algoritmos manipulam dados, quando estes dados estão organizados (dispostos) de forma coerente, caracterizam uma forma, uma estrutura de dados. São a organização e as operações que manipulam esta determinada estrutura que as identificam como únicas.



Deve-se sempre ter atenção à escolha de uma estrutura de dados, pois, quando apropriada, esta escolha pode tornar um problema complicado em um de solução trivial.

A escolha da forma de representar os dados e as regras de sua manipulação é, em geral, uma tarefa que não depende apenas dos recursos disponíveis, mas também das operações a serem realizadas sobre eles.



DESENVOLVIMENTO: UNIDADE I

1.1. Tipos de Dados

A noção de tipo de dados ocorre na maioria das linguagens de programação. Ao se declarar o tipo de uma variável, delimita-se o conjunto de valores que ela pode assumir e as operações que podem ser efetuadas com ela.

De uma maneira geral, uma linguagem oferece alguns tipos básicos predefinidos que chamamos Tipos Primitivos. Os tipos mais comuns a todas as linguagens são: inteiro, real, lógico e caractere.



Na linguagem Pascal os principais **tipos primitivos** de dados são: *integer*, *char*, *boolean*, *real*. Cada um destes tipos possui valores possíveis que podem assumir e operações permitidas.

Por exemplo, no tipo *integer*, os valores possíveis são os números inteiros (negativos, zero ou positivos) e as operações permitidas são: adição, subtração, multiplicação, divisão inteira, resto da divisão.

Novos tipos podem ser construídos a partir dos tipos primitivos. O formato geral é:

tipo nome_do_tipo = definição do tipo

Pode-se definir um tipo de dado simples, que são valores indivisíveis, como os tipos básicos *integer*, *boolean*, *real* e *char* do Pascal. Existe também o tipo de dado estruturado que definem uma coleção de valores simples.

No caso da linguagem Pascal os tipos podem ser colocados em três categorias: simples, estruturado e apontadores (ZIVIANE, 2005).



Exemplos de **tipos estruturados** (a-arranjo, b-registro, c-conjunto) definido através da linguagem Pascal.

a) **type** cartão = **array** [1..80] **of** char;

b) **type** aluno = **record**

nome: **string**;

idade: **integer**;

sexo: **char**;

end;

c) type conjint = set of 1..9;
--

1.2. Tipo Abstrato de Dados – TAD

Um tipo abstrato de dados pode ser visto como um modelo matemático, acompanhado das operações definidas sobre o modelo. O conjunto dos inteiros acompanhado das operações de adição, subtração e multiplicação forma um exemplo de um tipo abstrato de dados (ZIVIANE, 2005).

Para entendermos melhor sobre TAD vamos avaliar um pouco o que é abstração. A abstração permite, por exemplo, com que uma pessoa possa pensar sobre como se dirige um carro qualquer sem a necessidade de alguém lhe especifique uma marca ou modelo (pois todos os carros são dirigidos da mesma forma).

A variável de uma linguagem de programação é uma abstração, pois representa virtualmente na memória do computador determinado objeto que existe no mundo real. É só lembrar que utilizamos variáveis e suas operações sem nos preocuparmos como estas foram definidas e são armazenadas.

Os tipos abstratos de dados podem ser considerados generalizações de tipos primitivos de dados, da mesma forma que procedimentos são generalizações de operações primitivas tais como adição, subtração e multiplicação.

<i>E</i>	<p>Definição de um tipo abstrato de dados Lista de inteiros como:</p> <ul style="list-style-type: none"> ➔ O tipo: Type lista = array [1..10] of integer; ➔ Operações sobre o tipo definido: <ol style="list-style-type: none"> 1. faça a lista vazia; 2. inserir um elemento na lista; 3. pesquisar um elemento na lista; 4. remover o último elemento da lista.
----------	---

Podemos implementar tipos abstratos de dados usando basicamente dois recursos presente nas principais linguagens de programação:

- Vetor (quanto utilizarmos vetor na implementação de tipos abstratos, dizemos que a implementação utilizou alocação seqüencial).
- Alocação dinâmica de memória (quanto utilizarmos alocação dinâmica de memória na implementação de tipos abstratos, dizemos que a implementação utilizou alocação encadeada).

Iremos estudar tipos abstratos utilizando estes dois tipos de alocação dos dados.

1.3. Procedimentos e Funções

Em programação estruturada o procedimento e a função são necessários para generalizar as operações que manipulam/modificam as estruturas de dados.

O procedimento é uma espécie de subrotina que espera ou devolve valores sobre os argumentos de sua chamada e não retorna valor a partir do seu nome.

O Argumento pode ser uma variável ou uma constante que tem seu valor transferido para o parâmetro correspondente na chamada da função ou procedimento.

Os parâmetros são variáveis definidas no escopo da função ou do procedimento e que determinam os valores (tipos) esperados ou devolvidos de um procedimento ou função.

Normalmente o número de parâmetros definidos para um procedimento ou função deve ser igual ao número de argumentos na sua chamada.

O nome da variável parâmetro pode ser diferente do nome da variável que define o argumento.

Um parâmetro pode ser enviado para o procedimento ou função por valor ou referência. Quando o envio for por valor, o parâmetro da subrotina recebe o valor da variável correspondente na chamada. Quando o envio for por referência, o parâmetro da subrotina recebe o endereço da variável correspondente na chamada, conseqüentemente, tudo o que for alterado na variável na subrotina, será alterado na variável correspondente da chamada, o que não acontece no parâmetro por valor. Em Pascal os parâmetros por referência recebem a palavra reservada **var** antes do nome do parâmetro.

Assim como o procedimento, a função espera ou devolve valores sobre os argumentos que estão associados aos parâmetros quando a função é chamada no corpo do programa. Mas diferente do procedimento a função pode devolver um valor sobre o seu nome.



```
program exemplo_procedimento;  
  
procedure soma (a, b : integer; var c : integer);  
begin  
    c := a + b;  
end;  
  
var x, y, z: integer;  
begin
```

```
write ('Escreva um numero: '); readln (x);  
write ('Escreva outro numero: '); readln (y);  
soma (x, y, z);  
writeln ('Soma = ', z);  
soma (10, 20, z);  
writeln ('Soma = ', z);  
end.
```

E

```
program exemplo_funcao;  
function soma (a, b : integer) : integer;  
begin  
    soma := a + b;  
end;  
var x, y, z: integer;  
begin  
    write ('Escreva um numero: '); readln (x);  
    write ('Escreva outro numero: '); readln (y);  
    z := soma (x, y);  
    writeln ('Soma = ', z);  
    z := soma (10, 20);  
    writeln ('Soma = ', z);  
end.
```

1.4. Pilha Seqüencial

Algumas estruturas de dados disciplinam a forma como os elementos entram e saem destas estruturas. Pilhas e Filas são exemplos destas estruturas em que os elementos são inseridos e retirados de posições especiais.

No caso da **Pilha** a disciplina de acesso utilizada recebe o nome de LIFO (Last In First Out), ou seja, o último elemento a entrar na pilha é também o primeiro que será retirado no caso de uma remoção de um elemento desta estrutura. Todos os elementos na estrutura de dados Pilha são inseridos e retirados a partir de uma mesma posição denominada **Topo da pilha**.

A estrutura de implementação de uma pilha utilizando vetor pode ser observada na Figura 1.



Figura 1 - Estrutura de implementação da Pilha seqüencial

Supondo que o TAD pilha que estamos construindo realize as operações:

1. Iniciapilha: inicializa a estrutura pilha.
2. Empilha: inserir um novo elemento na pilha.
3. Desempilha: retirar o elemento do topo da pilha.

Vamos analisar como ficaria a implementação deste TAD. Inicialmente iremos declarar um tipo capaz de receber os valores desta pilha.

```
type pilha = record
    topo : integer;
    elementos : array [1..MAX] of char;
end;
```

Programas 1 - Declaração do tipo Pilha

O tipo declarado no Programas 1 é composto de dois campos:

- Topo: é do tipo inteiro, nele será armazenado o valor da posição do vetor onde o valor que está no topo se situa.
- Elementos: é um vetor de tamanho MAX que pode receber somente 1 caracter em cada posição. MAX pode ser uma constante que deve ser declarada antes da declaração do tipo.

```
procedure iniciapilha (var P : pilha);
begin
    P.topo := 0;
end;
```

Programas 2 - Procedimento iniciapilha

O procedimento *iniciapilha* recebe o parâmetro *P* por referência, que contém a nossa pilha, e atribui o valor 0 (zero) ao campo *topo* da pilha indicando que a pilha está vazia.

Esta é a operação de iniciar uma pilha

```
procedure empilha (var P : pilha; D : char);  
begin  
  if (P.topo = MAX)  
    then writeln ('Overflow')  
    else begin  
      inc (P.topo);  
      P.elementos [P.topo] := D;  
    end;  
end;
```

Programas 3 - Procedimento empilha

O procedimento *empilha* recebe o parâmetro *P* do tipo *pilha* por referência e o parâmetro *D* do tipo *char* por valor. O parâmetro *D* deverá conter o valor a ser inserido na pilha. Inicialmente é verificado se o *topo* da pilha é igual a *MAX* que é a constante com o tamanho da pilha, caso seja, não é possível a inserção de elementos nesta pilha. Neste caso o procedimento escreve a mensagem *Overflow* na tela e finaliza o seu trabalho.



A tentativa de inserir valores em uma pilha cheia é chamado de *Overflow* (estouro de pilha).

Caso o *topo* não seja igual a *MAX*, o procedimento irá incrementar o valor do campo *topo* da pilha e guardar no campo *elementos* da pilha na posição *topo* o valor da variável *D*.

Esta é a operação de empilhar um valor.

```
procedure desempilha (var P : pilha; var D : char);  
begin  
  if (P.topo = 0)  
    then writeln ('Underflow')  
    else begin  
      D := P.elementos [P.topo];  
      dec (P.topo);  
    end;  
end;
```

Programas 4 - Procedimento desempilha

O procedimento desempilha recebe o parâmetro P do tipo pilha e o parâmetro D do tipo char, ambos por referência. O parâmetro D deverá retornar com o valor retirado da pilha. Inicialmente é verificado se o topo da pilha é igual a zero, caso seja, não existe valor a ser desempilhado. Neste caso o procedimento escreve a mensagem *Underflow* na tela e finaliza o seu trabalho.



A tentativa de remover valores de uma pilha vazia é chamada de *Underflow*.

Caso o topo não seja igual a zero, o procedimento irá guardar na variável D o valor do campo elementos na posição topo e decrementar o topo.

Esta é a operação de desempilhar um valor.

Apesar da simplicidade de implementar as operações para manipular uma pilha, podemos observar a importância desta estrutura em diversas partes da computação.

A propriedade LIFO da pilha a torna uma ferramenta ideal para processamento de estruturas aninhadas de profundidade imprevisível, situação em que é necessário garantir que subestruturas mais internas sejam processadas antes da estrutura que as controlam. Estruturas aninhadas ocorrem freqüentemente na prática. Um exemplo simples é a situação em que é necessário caminhar em um conjunto de dados e guardar uma lista de coisas a fazer posteriormente. As pilhas ocorrem também em natureza recursiva, tais como as árvores, assim como são utilizadas para implementar a recursividade (ZIVIANE, 2005).

1.5. Fila seqüencial

Para a Fila a disciplina de acesso utilizada é a FIFO (First In First Out), o primeiro elemento que entra na fila será também o primeiro elemento que sairá da fila. A posição nesta estrutura por onde os elementos são inseridos recebe o nome de **Fim da Fila**, e a posição por onde os elementos são retirados desta estrutura recebe o nome de **Começo da Fila**.

A diferença básica entre a pilha e a fila é que na pilha temos uma única extremidade por onde os elementos são inseridos e retirados (topo da pilha). No caso da fila os elementos entram por uma extremidade (fim da fila) e saem por outra (começo da fila).

A estrutura de implementação de uma fila utilizando vetor pode ser observada na Figura 2.

Fila

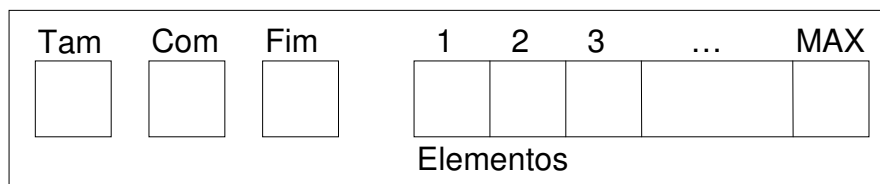


Figura 2 - Estrutura de implementação da Fila seqüencial

Para a adição de um elemento, move-se o ponteiro Fim e para a retirada, move-se o ponteiro Com.

Após qualquer operação, devemos ter sempre o ponteiro Com indicando o início de fila e o ponteiro Fim indicando o fim da fila. Isto implica na movimentação de Com e Fim quando de uma remoção ou inserção respectivamente como mostrado na Figura 3.

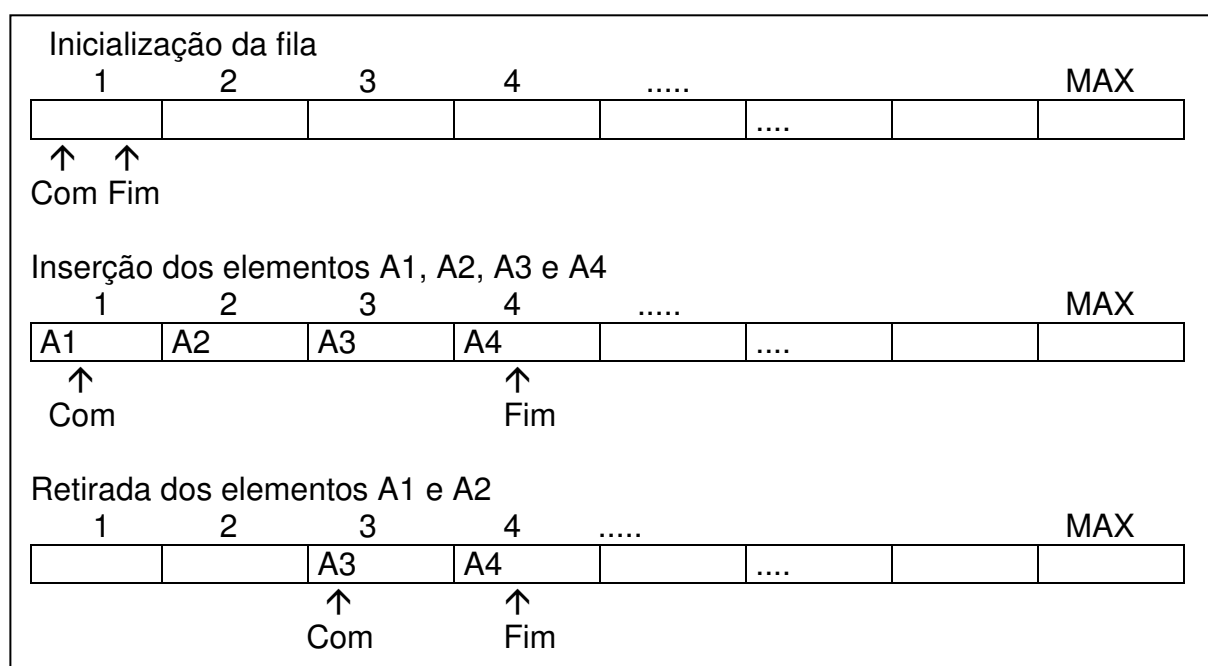


Figura 3 - Funcionamento da Fila

A movimentação dos ponteiros provoca uma falsa impressão de memória esgotada, ou seja, que em determinado instante não terá como inserir mais nenhum elemento. Para que isto não ocorra consideremos que os elementos estejam alocados como se estivessem em círculo conforme Figura 4. Desta forma a fila passa a se chamar Fila Circular e os elementos da fila se encontram em posições contíguas de memória, em algum lugar do círculo, delimitado pelos apontadores

Com e Fim. Considerando todos estes aspectos a definição de um tipo fila pode ser declarada como na Figura 4.

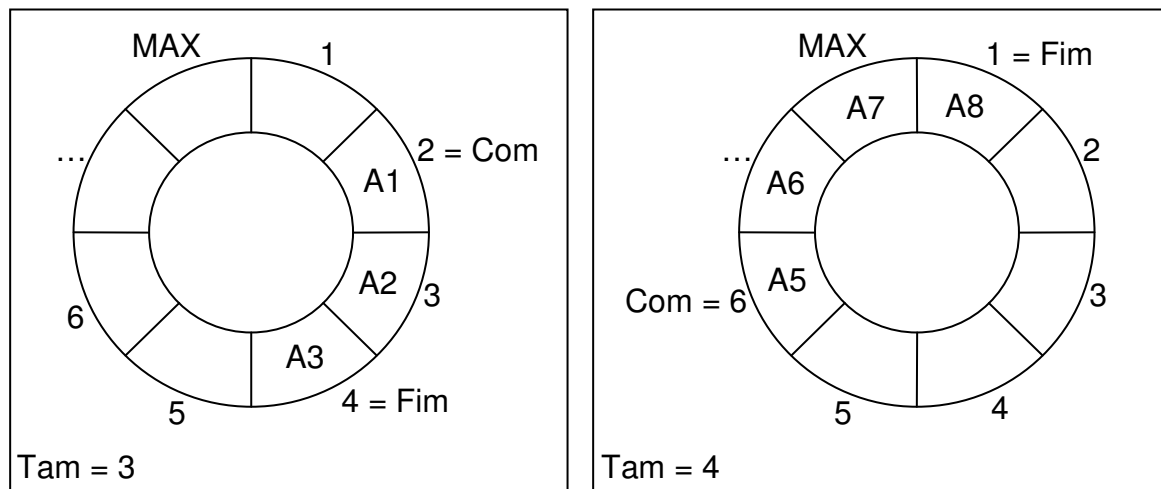


Figura 4 - Dois momentos distintos de elementos alocados em uma Fila Circular

Supondo que o TAD fila que estamos construindo realize as operações:

1. iniciafila: inicializa a estrutura fila.
2. inserefila: inserir um novo elemento no fim da fila.
3. retirafile: retirar o elemento do início da fila.

Vamos analisar como ficaria a implementação deste TAD. Inicialmente iremos declarar um tipo capaz de receber os valores desta fila.

```
type fila = record
    tam : integer;
    com : integer;
    fim : integer;
    elementos : array [1..MAX] of string;
end;
```

Programas 5 - Declaração do tipo Fila

O tipo declarado no Programas 5 é composto de três campos:

- tam: é do tipo inteiro, nele será contado o número de valores da fila.
- com: é do tipo inteiro, nele será armazenado a posição do vetor onde começa a fila.
- fim: é do tipo inteiro, nele será armazenado a posição do vetor onde a fila termina.

- elementos: é um vetor de tamanho MAX que pode receber um conjunto de caracteres em cada posição. MAX pode ser uma constante que deve ser declarada antes da declaração do tipo.

```
procedure iniciafila (var F : fila);  
begin  
    F.com := 1;  
    F.fim := 0;  
    F.tam := 0;  
end;
```

Programas 6 - Procedimento iniciafila

O procedimento iniciafila recebe o parâmetro F por referência, que contém a nossa fila, atribui o valor 1 ao começo e 0 (zero) aos campos fim e tam da fila indicando que ela está vazia.

Esta é a operação de iniciar uma fila

```
procedure inserefila (var F : fila; D : string);  
begin  
    if (F.tam = MAX)  
        then writeln ('Fila Cheia')  
    else begin  
        inc (F.tam);  
        F.fim := F.fim mod MAX + 1;  
        F.elementos [F.fim] := D;  
    end;  
end;
```

Programas 7 - Procedimento inserefila

O procedimento inserefila recebe o parâmetro F do tipo fila por referência e o parâmetro D do tipo string por valor. O parâmetro D deverá conter o valor a ser inserido na fila. Inicialmente é verificado se o tamanho da fila é igual a MAX que é a constante com o tamanho da fila, caso seja, não é possível a inserção de elementos nesta fila. Neste caso o procedimento escreve a mensagem Fila Cheia na tela e finaliza o seu trabalho.

Caso o tamanho da fila não seja igual a MAX, o procedimento irá incrementar o valor do campo tam da fila. Depois, para controlar o andar do campo Fim, fazer o incremento de 1 usando a função MOD (resto da divisão). Assim, quando este campo chegar à posição Máxima do vetor declarada, neste caso a constante MAX, e

tiver que ser realizada a inserção de um novo elemento, o ponteiro poderá passar do índice de MAX para 1, como mostra abaixo:



$F.fim := F.fim \bmod MAX + 1;$

Considere que o valor de MAX é 100, assim se queremos passar da posição 1 para a posição 2 fazemos $(1 \bmod 100) + 1 = 2$, e assim sucessivamente. Para passar da posição 100 para a posição 1 será $(100 \bmod 100) + 1$.

Por fim, o procedimento irá guardar no campo elementos da fila na posição fim o valor da variável D.

Esta é a operação de inserir um valor na fila.

```
procedure retirafila (var F : fila; var D : string);
begin
  if (F.tam = 0)
  then writeln ('Fila vazia')
  else begin
    dec (F.tam);
    D := F.elementos [F.com];
    F.com := F.com mod MAX + 1;
  end;
end;
```

Programas 8 - Procedimento retirafila

O procedimento retirafila recebe os parâmetros F do tipo fila e D do tipo string, ambos passados por referência. O parâmetro D deverá retornar com o valor retirado da fila. Inicialmente é verificado se o tamanho da fila é igual a 0 (zero), caso seja, não é possível remover elementos nesta fila. Neste caso o procedimento escreve a mensagem Fila vazia na tela e finaliza o seu trabalho.

Caso contrário, o procedimento irá decrementar o tamanho da fila, guardar na variável D o valor do campo elementos na posição com da fila e incrementar o começo seguindo a mesma lógica da inserção. Para controlar o andar do campo com, fazer o incremento de 1 usando a função MOD (resto da divisão). Assim, quando este campo chegar à posição Máxima do vetor declarada, neste caso a constante MAX, e tiver que ser realizada a remoção de um novo elemento, o ponteiro poderá passar do índice de MAX para 1, como mostra abaixo:

F

$F.com := F.com \bmod MAX + 1$;
 Considere que o valor de MAX é 100, assim se queremos passar da posição 1 para a posição 2 fazemos $(1 \bmod 100) + 1 = 2$, e assim sucessivamente. Para passar da posição 100 para a posição 1 será $(100 \bmod 100) + 1$.

Esta é a operação de remover um valor da fila.

Filas são utilizadas quando desejamos processar itens de acordo com a ordem “primeiro-que-chega, primeiro-atendido”. Sistemas operacionais utilizam filas para regular a ordem na qual, tarefas devem receber processamento e recursos devem ser alocados a processos.

1.6. Lista Linear seqüencial

É a estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem linear que existe entres estes dados. Uma lista linear é composta de nós, os quais podem conter cada um deles, um dado primitivo ou um dado composto.

Define-se uma lista linear como sendo o conjunto de $n \geq 0$ de nós x_1, x_2, \dots, x_n , organizados estruturalmente de forma a refletir as posições relativas dos mesmos. Se $n > 0$ então x_1 é o primeiro nó; para $1 < k < n$, o nó k é precedido pelo nó x_{k-1} e é seguido do nó x_{k+1} ; e x_n é o último nó. Quando $n = 0$, dizemos que a lista é vazia.

Podemos implementar listas lineares seqüenciais de duas formas:

1. Por contigüidade dos nós: usando o construtor de tipo vetor.
2. Por encadeamento dos nós: usando vetor.

1.6.1. Lista Linear seqüencial por contigüidade dos nós

A representação por contigüidade explora a seqüencialidade da memória do computador de tal forma que os nós de uma lista sejam armazenados em endereços contíguos, ou igualmente distanciados uns dos outros.

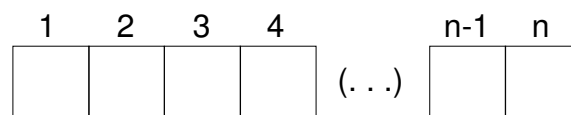


Figura 5 - Estrutura de lista linear por contigüidade dos nós

Supondo que o TAD lista que estamos construindo realize as operações:

1. Inicializar uma lista

Nesta operação devemos percorrer toda lista atribuindo um valor inicial para os seus elementos a fim de limpar a lista. Devemos também atribuir um valor inicial para a variável responsável pela marcação do fim da lista.

2. Inserir um nó na lista

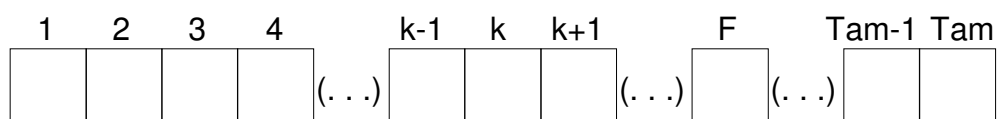


Figura 6 - Inserção de um nó na lista

Se existir espaço disponível na lista, devemos procurar a posição em que o novo elemento deve ser inserido, seja k esta posição. A posição onde um novo elemento será inserido depende do critério de ordem escolhido para lista (ordem alfabética, por exemplo), então mover o elemento k para posição $k+1$, o elemento $k+1$ para posição $k+2$, e assim por diante. Ou seja, devemos gerar espaço para inserção do novo elemento.

Para que esta operação preserve os dados já armazenados na lista é necessário que esta operação de deslocamento (shift) seja executada do fim da lista para a posição k .

3. Remover um nó da lista

Devemos procurar a posição onde se encontra o elemento a ser removido, seja k esta posição. Se encontrarmos este elemento então devemos mover o valor contido na posição $k+1$ para a posição k , $k+2$ para posição $k+1$, até o fim da lista. Ou seja, esta operação deve comprimir a lista sobre o elemento a ser removido.

Vamos analisar como ficaria a implementação deste TAD. Inicialmente iremos declarar um tipo capaz de receber os valores desta lista.

```
type lista = record
    fim : integer;
    elementos : array [1..MAX] of string;
end;
```

Programas 9 - Declaração do tipo Lista

O tipo declarado no Programas 9 é composto de dois campos:

- fim: é do tipo inteiro, nele será armazenado a posição do vetor onde a lista termina.
- elementos: é um vetor de tamanho MAX que pode receber um conjunto de caracteres em cada posição. MAX pode ser uma constante que deve ser declarada antes da declaração do tipo.

```
procedure inicialista (var L:lista);  
var i: integer;  
begin  
  for i := 1 to MAX do  
    L.elementos[i] := "";  
  L.fim := 0;  
end;
```

Programas 10 - Procedimento inicialista

O procedimento inicialista recebe o parâmetro L por referência, que contém a nossa lista, preenche todas as posições do campo elementos com um caracter vazio e guarda o valor 0 (zero) na posição fim indicando que lista está vazia.

Esta é a operação de iniciar uma lista seqüencial por contigüidade dos nós.

```
procedure inserelista (var L:lista; n:string);  
var i, p : integer;  
begin  
  if L.fim = MAX then  
    write ('Lista cheia')  
  else begin  
    i := 1;  
    while ((L.elementos[i] <> "") and (n > L.elementos[i])) do  
      inc (i);  
    p := i;  
    i := L.fim;  
    while (i <> p-1) do  
      begin  
        L.elementos[i+1] := L.elementos[i];  
        dec (i);  
      end;  
    inc (L.fim);  
    L.elementos[p] := n;  
  end;  
end;
```

Programas 11 - Procedimento inserelista

O procedimento `inserelista` recebe o parâmetro `L` do tipo `lista` por referência e o parâmetro `n` do tipo `string` por valor. O parâmetro `n` deverá conter o valor a ser inserido na lista. As variáveis “`i`” e “`p`” são utilizadas como auxiliares no procedimento.

Inicialmente é verificado se o fim da lista é igual a `MAX` que é a constante com o tamanho da lista, caso seja, não é possível a inserção de elementos nesta lista. Neste caso o procedimento escreve a mensagem `Lista cheia` na tela e finaliza o seu trabalho.

Caso o fim da lista não seja igual a `MAX`, o procedimento irá iniciar a variável `i` com o valor 1 e entrar em um loop enquanto os valores das posições do vetor de elementos forem diferentes de vazio e o valor a ser inserido for maior que o valor destas posições. Dentro deste loop está somente a variável `i` que a cada iteração incrementa o seu valor objetivando passar pelas posições do vetor enquanto as condições forem satisfeitas. Ao sair do loop a variável `i` possui a posição de inserção do valor na lista, com isso, a variável auxiliar `p` recebe o valor da variável `i`, e a variável `i` recebe o fim da lista. O novo loop está movendo os valores do vetor uma posição à frente para que o valor possa ser inserido na posição correta obedecendo a ordem da lista. O loop acontece enquanto a variável `i` for diferente de `p-1` e as ações dentro do loop são: copiar o valor para a posição a frente, seguindo do último para a posição a ser inserida e decrementando o `i` que indica a posição a ser copiada a frente. Findado o loop, a posição a ser inserida já poderá receber o valor enviado. O próximo passo é incrementar o fim e guardar o valor a ser inserido. Este valor foi enviado no parâmetro `n` que será copiado para a lista, campo `elementos`, posição `p`.

Esta é a operação de inserir um valor na lista seqüencial por contigüidade dos nós.

```
procedure removelista (var L:lista; n:string);
var i, p : integer;
begin
  if L.fim = 0 then
    write ('Lista vazia')
  else begin
    i := 1;
    while ((i <= L.fim) and (L.elementos[i] <> n)) do
      inc (i);
    if i > L.fim then
      write ('Nome nao encontrado')
    else begin
      p := i;
      while (i <> L.fim) do
        begin
          L.elementos[i] := L.elementos[i+1];
```

```
        inc (i);  
    end;  
    L.elementos[L.fim] := "  
    dec (L.fim);  
end;  
end;  
end;
```

Programas 12 - Procedimento removelista

O procedimento removelista recebe o parâmetro L do tipo lista por referência e o parâmetro n do tipo string por valor. O parâmetro n deverá conter o valor a ser removido da lista. As variáveis “i” e “p” são utilizadas como auxiliares no procedimento.

Inicialmente é verificado se o fim da lista é igual a 0 (zero), caso seja, não é possível a remoção de elementos desta lista. Neste caso o procedimento escreve a mensagem Lista vazia na tela e finaliza o seu trabalho.

Caso o fim da lista não seja igual a 0 (zero), o procedimento irá iniciar a variável i com o valor 1 e entrar em um loop enquanto a variável i for menor ou igual ao fim e os valores das posições do vetor de elementos forem diferentes do valor a ser inserido. Dentro deste loop está somente a variável i que a cada iteração incrementa o seu valor objetivando passar pelas posições do vetor enquanto as condições forem satisfeitas. Ao sair do loop, caso a variável i for maior que o fim, significa que o valor a ser removido foi comparado com todos do vetor, mas não foi encontrado, neste caso o procedimento escreve a mensagem nome não encontrado na tela e finaliza o seu trabalho. Caso contrário, a variável i possui a posição de remoção do valor na lista, com isso, a variável auxiliar p recebe o valor da variável i. O novo loop está movendo os valores do vetor uma posição para traz sobrepondo o valor a ser removido. O loop acontece enquanto a variável i não chegar ao fim e as ações dentro do loop são: copiar o valor da frente para a posição de traz, seguindo do valor a ser removido para a última posição e incrementando o i que indica a posição a ser sobreposta. Findado o loop, a posição a ser removida já foi substituída. O próximo passo é limpar a antiga última posição do vetor e decrementar o fim.

Esta é a operação de remover um valor na lista seqüencial por contigüidade dos nós.

No caso da representação de listas lineares por contigüidade dos nós, dois problemas foram detectados:

1. O número máximo de elementos fica limitado ao tamanho do vetor.
2. Algumas operações primitivas comprometem o desempenho da lista, pois implicam num grande esforço computacional para cumprir suas especificações. Por exemplo, para inserção de um elemento na primeira posição de uma lista com 2.000 elementos armazenados. Nesta situação

são necessárias 2.000 atribuições para gerar espaço para o novo primeiro elemento da lista.

Para resolver o segundo problema (complexidade das operações de inserção e remoção), podemos usar a técnica de encadeamento. A técnica de encadeamento pode ser implementada usando-se o vetor. Mas neste caso continuamos com o primeiro problema, a estrutura de lista continua limitada para o máximo número de elementos do vetor utilizado na sua declaração

Duas novas implementações da estrutura de dados lista linear será apresentada:

1. Lista Linear Encadeada usando Vetor
2. Lista Linear Encadeada usando Alocação Dinâmica de Memória.

A lista linear encadeada usando vetor será apresentada logo abaixo e a encadeada usando alocação dinâmica de memória estudaremos mais a frente.

1.6.2. Lista Linear encadeada usando vetor

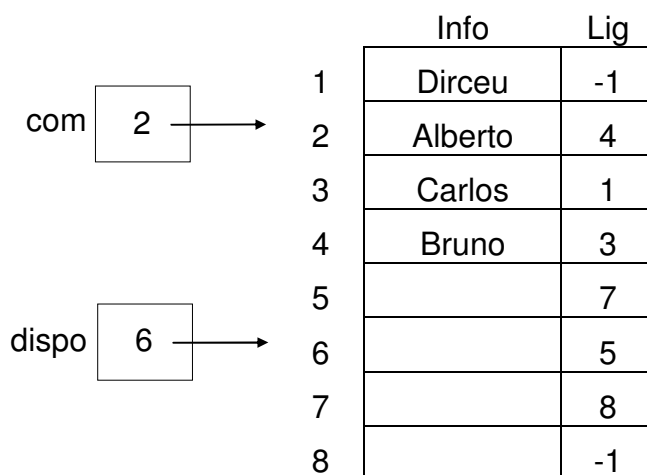


Figura 7 - Estrutura da lista linear encadeada usando vetor

Cada elemento da lista é um registro com dois campos: info (informação) utilizado para armazenar os valores da lista, e o campo lig (ligação) utilizado para prover o encadeamento da lista. Cada elemento indica qual é o elemento que vem em seguida através do valor no campo lig. O valor -1 marca o fim da lista. O começo da lista de posições disponíveis da estrutura está indicada na variável dispo, e o começo da lista propriamente dita está indicado na variável com.

Para esta nova representação o processo de inserção e remoção de nós na lista é simplificado pelo simples acerto de ligações.

Supondo que o TAD lista que estamos construindo realize as operações:

1. Inicializar uma lista

Nesta operação devemos percorrer toda lista atribuindo um valor inicial para os seus elementos a fim de limpar a lista. Devemos também atribuir um valor inicial para as variáveis de controle com, dispo além de preencher o campo lig da estrutura indicando a próxima posição.

2. Inserir um nó na lista

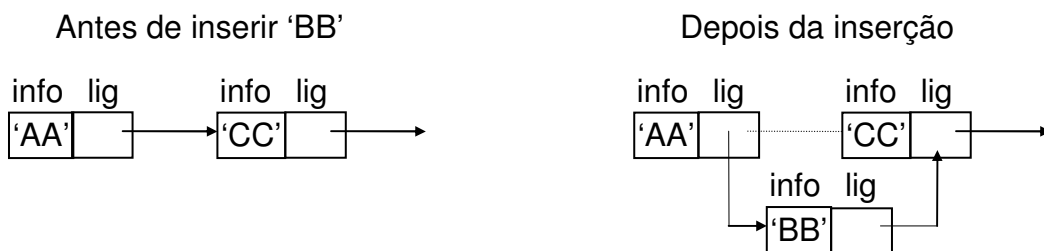


Figura 8 - Processo de inserção de um nó na lista

3. Remover um nó na lista

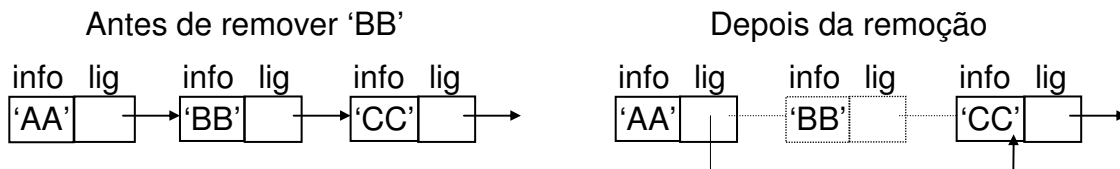


Figura 9 - Processo de remoção de um nó da lista

Vamos analisar como ficaria a implementação deste TAD. Inicialmente iremos declarar um tipo capaz de receber os valores desta lista.

```
type no = record
    info : char;
    lig : integer;
end;
type lista = array [1..MAX] of no;
```

Programas 13 - Declaração do tipo Lista encadeada

Para implementação da lista encadeada declaramos dois tipos: `no` e `lista`. O tipo `no` é composto de dois campos:

- `info`: é do tipo `char`, nele será armazenado os valores da lista.
- `lig`: é do tipo inteiro, ele irá indicar quem é o próximo valor depois da posição atual.

O tipo `lista` é um vetor de tamanho `MAX` onde cada posição tem o formato de `no`, ou seja, cada posição é composta pelos campos `info` e `lig`. `MAX` pode ser uma constante que deve ser declarada antes da declaração do tipo.

```
procedure inicialista (var L: lista; var c, d:integer);
var i: integer;
begin
  for i := 1 to MAX do
    begin
      L[i].info := ' ';
      L[i].lig := i+1;
    end;
  L[MAX].lig := -1;
  c := -1;
  d := 1;
end;
```

Programas 14 - Procedimento inicialista

O procedimento `inicialista` recebe o parâmetro `L` por referência, que contém a nossa lista. Os parâmetros `c` e `d` correspondem a **com** e **dispo** respectivamente que também foram passados por referência. O procedimento possui uma variável `i` auxiliar para percorrer a lista guardando no campo `info` um valor vazio e no campo `lig` o valor da posição atual mais um, indicando que este campo está ligado ao próximo. Findado o loop, a última posição do vetor na posição `lig` recebe -1 indicando fim, o começo recebe -1 indicando vazio e o disponível recebe 1 indicando que a primeira posição é a próxima disponível para inserção.

Esta é a operação de iniciar uma lista seqüencial encadeada no vetor.

```
procedure inserelista (var L:lista; var c, d:integer; n:char);
var aux, ant : integer;
begin
  if d = -1
  then write ('Lista cheia')
  else begin
    aux := c;
```

```
ant := -1;
while (L[aux].info<=n) and (aux<>-1) do
begin
    ant := aux;
    aux := L[aux].lig;
end;
aux := d;
d := L[d].lig;
L[aux].info := n;
if ant = -1 then
begin
    L[aux].lig := c;
    c := aux;
end
else begin
    L[aux].lig := L[ant].lig;
    L[ant].lig := aux;
end;
end;
end;
```

Loop que encontra a posição onde o valor será inserido na lista

Esta condição será executada quando a posição do valor a ser inserido for a primeira

Esta condição será executada quando a posição do valor a ser inserido for após o primeiro valor da lista

Programas 15 - Procedimento inserelista

O procedimento *inserelista* recebe o parâmetro *L* por referência, que contém a nossa lista. Os parâmetros **c** e **d** correspondem a **com** e **dispo** respectivamente que também foram passados por referência e a variável *n* que terá o valor a ser inserido. No procedimento são declaradas as variáveis **aux** e **ant** as quais são auxiliares no posicionamento dos valores. Inicialmente verifica-se a existência de posições para inserção, testando se a variável *d* = -1, caso seja, o procedimento escreve a mensagem “Lista cheia” e encerra o seu trabalho. Caso contrário, a variável *aux* recebe o começo e a variável *ant* recebe -1. Posteriormente, elas entram em um loop onde o objetivo é posicionar a variável *aux* na posição posterior a do valor que será inserido e a variável *ant* na posição anterior deste mesmo valor, para isso, o loop acontece enquanto o valor da lista, o qual satisfeito as condições irá passar por todos, for menor ou igual ao valor a inserir e *aux* diferente de -1 indicando que chegou ao fim.

Findado o loop, a variável *aux* recebe a posição disponível e o disponível recebe a próxima posição disponível. Depois, o valor é inserido na posição disponível da lista. Inserido o valor, a atualização do campo de ligação da lista segue dois critérios: inserção na primeira posição, onde o valor da variável *ant* será -1, neste caso a posição do valor inserido irá ligar no começo e ela passará a ser o novo começo; inserção nas próximas posições onde o campo *lig* da posição inserida recebe o *lig* da posição anterior e o *lig* da anterior recebe a posição inserida.

Esta é a operação de inserir um valor na lista seqüencial encadeada no vetor.

```
procedure removelista (var l:lista; var c, d:integer; n:char);
var aux, ant: integer;
begin
  if c = -1
  then write ('Lista vazia')
  else begin
    aux := c;
    ant := -1;
    while (aux <> -1) and (l[aux].info<>n) do
    begin
      ant := aux;
      aux := l[aux].lig;
    end;
    if aux = -1
    then write ('Nome nao encontrado')
    else begin
      if aux = c then
        c := l[c].lig;
      else l[ant].lig := l[aux].lig;
      l[aux].lig := d;
      l[aux].info := ' ';
      d := aux;
    end;
  end;
end;
```

Loop que encontra a posição onde o valor será inserido na lista

Remoção no começo

Remoção após o primeiro valor da lista

Programas 16 - Procedimento removelista

O procedimento removelista recebe o parâmetro L por referência, que contém a nossa lista. Os parâmetros **c** e **d** correspondem a **com** e **dispo** respectivamente que também foram passados por referência e a variável n que terá o valor a ser removido. No procedimento são declaradas as variáveis **aux** e **ant** as quais são auxiliares no posicionamento dos valores. Inicialmente é verificado se existe valores na lista, testando se a variável c = -1, caso seja, o procedimento escreve a mensagem “Lista vazia” e encerra o seu trabalho. Caso contrário, a variável aux recebe o começo e a variável ant recebe -1. Posteriormente, elas entram em um loop onde o objetivo é posicionar a variável aux na posição do valor que será removido e a variável ant na posição anterior deste mesmo valor, para isso, o loop acontece enquanto o valor da lista, o qual satisfeito as condições irá passar por todos, for diferente do valor a ser removido e aux diferente de -1 indicando que chegou ao fim.

Findado o loop, é realizado um teste na variável aux, caso ela será igual a -1 indica que o valor a ser removido não está na lista, portanto o procedimento escreve

a mensagem “Nome não encontrado” e finaliza seu trabalho. Caso contrário, o valor deverá ser removido, porém a atualização do campo lig segue dois critérios: remoção da primeira posição, neste caso o começo deverá ser o campo lig do antigo começo; remoção de valores posteriores, o lig do valor anterior recebe o lig do valor a ser removido. Posteriormente, o lig da posição do valor a ser removido é ligada na posição disponível, o valor removido é limpo da lista e a posição do valor removido passa a ser a nova disponível.

Esta é a operação de remover um valor na lista seqüencial encadeada no vetor.



DESENVOLVIMENTO: UNIDADE II

2.1. Tipo Apontador

Um das características mais marcantes de linguagens estruturadas é permitir a criação e destruição de variáveis durante a execução do programa. O uso dessas variáveis possibilita a implementação das estruturas de dados dinâmicas. Essas variáveis criadas e destruídas durante a execução do programa são chamadas variáveis dinâmicas. Uma variável dinâmica não é declarada na parte de declaração de variáveis porque esta ainda não existe antes do seu tempo de execução, ela não possui sequer um nome, ficando a cargo dos ponteiros desempenharem esta função de “nome”. Uma variável dinâmica é sempre referenciada indiretamente por um apontador, ou seja, para cada variável dinâmica criada deve existir um apontador, que literalmente aponta para ela, permitindo a sua manipulação.

Os apontadores são declarados como as demais variáveis, seguindo a sintaxe da linguagem de programação utilizada. Eles são variáveis que armazenam o endereço de memória de outras variáveis, funcionando assim, como meio de referenciar uma variável dinâmica, permitindo o acesso a essa variável. Outra característica dos apontadores é que na sua declaração deve ser indicado qual o tipo de variável este irá armazenar o endereço.

A principal função dos ponteiros é permitir a criação e a manipulação das variáveis dinâmicas, as quais irão compor as estruturas de dados dinâmicas.

2.2. Ponteiros

2.2.1. Declaração


A sintaxe de declaração de um tipo ponteiro é a seguinte.



```
type identificador = ^tipo;  
{o símbolo ^ indica que o identificador é um ponteiro}
```


Isso indica que teremos um ponteiro com o nome de identificador com a capacidade de armazenar o endereço de memória de uma variável desse tipo.

Como os ponteiros são utilizados para implementar as estruturas de dados dinâmicas, o mais comum em Pascal é que seja criado um tipo do ponteiro desejado (type) e depois declaramos (var) as variáveis ponteiro deste tipo.

	<pre>type Nome = string[35]; PontNome = ^Nome; var nome1 : PontNome; {o conteúdo apontado por nome1 é do tipo string de tamanho 35}</pre>
---	--


2.2.2. Inicialização


Como os ponteiros armazenam endereços de memória, assim que eles são declarados o seu conteúdo é um valor desconhecido (como as demais variáveis), ou seja, não podemos prever para qual área de memória este está apontando, por isso, linguagens estruturadas permitem que um ponteiro seja inicializado, da seguinte forma:

	identificador := nil; {nil é um valor nulo e pode ser armazenado em uma variável do tipo apontador para indicar que a variável não contém endereço de memória de nenhuma variável dinâmica }
---	--

2.2.3. Criação de uma variável dinâmica

Para criação de uma variável dinâmica é necessário que exista uma variável do tipo apontador para o tipo da variável que se deseja criar, cuja sintaxe é a seguinte:

	new (identificador); {onde, identificador deve ser uma variável do tipo ponteiro para o tipo de variável a ser criada}
---	--

	Quando uma variável dinâmica é criada através da função NEW, um espaço de memória do tamanho da variável criada é alocado e este espaço ficará alocado até ser liberado.
---	--

2.2.4. Destruição de uma variável dinâmica

Para que o espaço de memória seja liberado é necessário que a variável dinâmica seja destruída, utilizando a seguinte sintaxe:



dispose (identificador);

2.2.5. Referência a uma variável dinâmica

Quando uma variável dinâmica é criada através de um ponteiro, este pode ser utilizado para referenciá-la, permitindo a manipulação (leitura, atribuição, escrita, etc) desta variável como outra variável qualquer, observe o exemplo:



```
program ManipulacaoVariaveisDinamicas;
uses crt;
type
  nome = string[30];
  ponteiro = ^nome;
var
  p1, p2 : ponteiro;
begin
  1 new(p1);
  2 new(p2);
  3 readln(p1^);
  4 p2^ := 'José Maria';
  5 p1^ := p1^ + ' ' + p2^;
  6 writeln(p1^);
  7 dispose(p1);
  8 dispose(p2);
end.
```

As variáveis p1 e p2 são variáveis ponteiro. O tipo ponteiro declarado aponta para endereço de nome e nome é um tipo string de tamanho 30, portanto o endereço de memória apontado pelas variáveis p1 e p2 pode receber valores string de tamanho máximo 30 caracteres.

No programa acima, é realizada duas alocações dinâmicas de memória, a primeira é apontada pela variável p1 e a segunda pela variável p2. O valor digitado pelo usuário é armazenado no endereço apontado pela variável p1 e no endereço

apontado pela variável p2 é armazenado a string 'José Maria'. O conteúdo do endereço apontado pela variável p1 é concatenado com um espaço e concatenado também com o conteúdo do endereço apontado pela variável p2 e armazenado na variável p1. O conteúdo do endereço de memória apontado por p1 é escrito na tela. Com o comando `dispose`, o endereço de memória apontado pela variável p1 e p2 é devolvido ao sistema operacional para ser utilizado em outra operação.

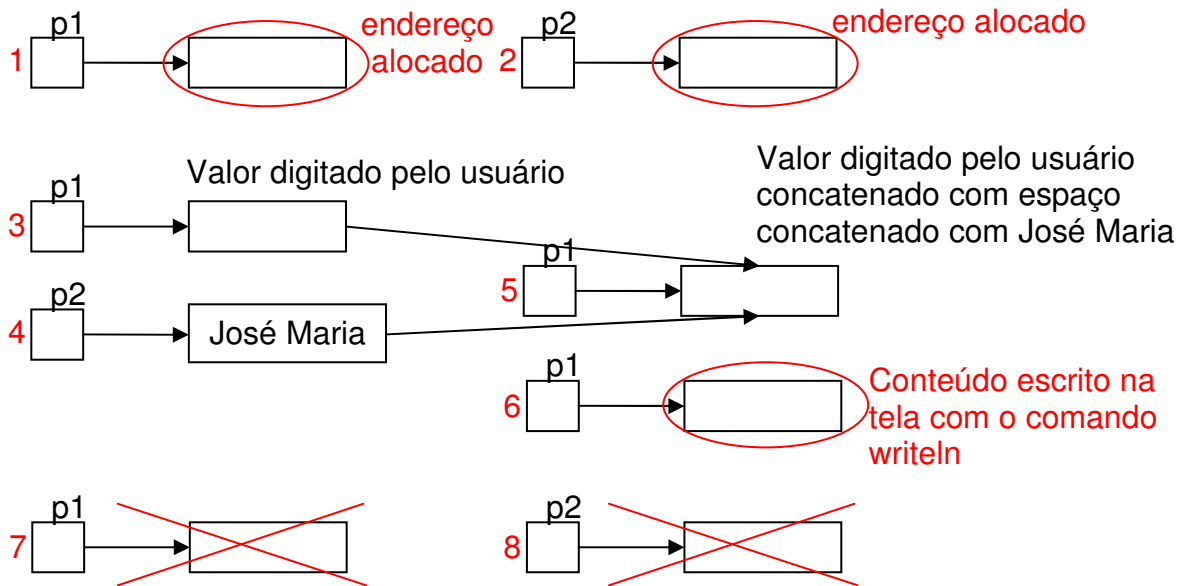



Figura 10 - Explicação do exemplo de alocação dinâmica de memória

2.3. Pilha usando alocação dinâmica de memória

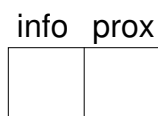
Um problema que normalmente acontece com as estruturas que usam do construtor de tipo vetor é a limitação do número máximo de elementos que podemos ter. Para aplicações em que necessitamos da estrutura de pilha, mas não podemos prever a quantidade de elementos que teremos, precisamos usar duas técnicas:

1. Alocação dinâmica de memória e
2. Encadeamento

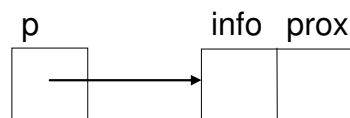
	<pre> program testa_alocacao_dinamica; type apontno = ^no; no = record info : char; prox : apontno; end; </pre>
---	---

```
var p,q : apontno;
    c : char;
begin
    q := nil;
    readln (c);
    new (p);
    p^.info := c;
    p^.prox := q;
    q := p;
    writeln (q^.info);
    dispose(p);
    readln;
end.
```

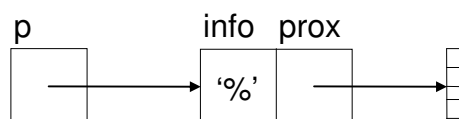
1. Na definição do tipo **apontno** estamos criando um tipo de variável para guardar endereço de estruturas **no**. A estrutura **no** é composta de dois campos, o campo **info** é o campo responsável por manter a informação na pilha e o campo **prox** é o campo responsável por possibilitar o encadeamento das informações na pilha.



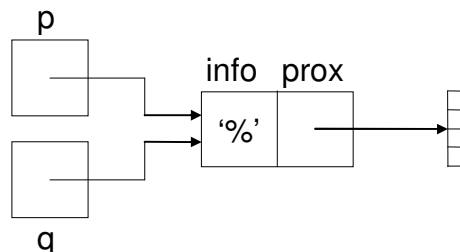
2. A atribuição $q := \text{nil}$; guarda na variável q um endereço nulo de memória.
3. A chamada da rotina de alocação dinâmica de memória $\text{new}(p)$; causa a criação de uma estrutura nó para onde a variável ponteiro p passa a apontar:



4. As atribuições $p^{\wedge}.\text{info} := c$; e $p^{\wedge}.\text{prox} := q$; guardam nos campos **info** e **prox** da estrutura nó apontada por p os valores guardados nas variáveis c e q respectivamente.



5. A atribuição $q := p$; faz com que as variáveis q e p contenham o mesmo endereço, i.e., os ponteiros q e p passam a apontar para a mesma estrutura na memória.



6. O comando de escrita `writeln (q^.info)`; apresenta o valor guardado no campo `info` da estrutura apontada por q que é também o valor no campo `info` da estrutura apontada por p pois ambos apontadores apontam para a mesma estrutura na memória.

Vamos analisar como ficaria a implementação da pilha utilizando alocação dinâmica de memória. Inicialmente iremos declarar um tipo capaz de receber os valores deste TAD.

```
type apontno = ^no;
  no = record
    info : char;
    prox : apontno;
  end;
```

Programas 17 - Declaração do tipo `apontno` e `no`

Para implementação da pilha declaramos dois tipos: `apontno` e `no`. O tipo `no` é composto de dois campos:

- `info`: é do tipo `char`, nele será armazenado os valores da pilha.
- `prox`: é do tipo `apontno`, ele irá indicar quem é o próximo no depois do atual.

O tipo `apontno` é do tipo endereço de `no`, ou seja, ele é um ponteiro capaz de guardar endereço de memória que tenha o formato do tipo `no`. Observe também que o campo `prox` do tipo `no` também é do tipo `apontno`, este fato justifica-se na utilização deste campo para realização do encadeamento proposto na criação da pilha encadeada por alocação dinâmica de memória.

```
procedure iniciapilha (var P : apontno);  
begin  
    P := nil;  
end;
```

Programas 18 - Procedimento iniciapilha

O procedimento iniciapilha recebe o parâmetro P por referência, que contém a nossa pilha. A atribuição do valor nil para esta variável indica que não temos nenhum valor na pilha atualmente.

Esta é a operação de iniciar uma pilha encadeada por alocação dinâmica de memória.

```
procedure empilha (var P : apontno; D : char);  
var aux : apontno;  
begin  
    new (aux);  
    if (aux = nil)  
        then writeln ('Overflow')  
        else begin  
            aux^.info := D;  
            aux^.prox := P;  
            P := aux;  
        end;  
end;
```

Programas 19 - Procedimento empilha

O procedimento empilha recebe o parâmetro P do tipo apontno por referência e o parâmetro D do tipo char por valor. O parâmetro D deverá conter o valor a ser inserido na pilha. É declarado no procedimento uma variável auxiliar do tipo apontno que será utilizada na criação da estrutura.

Inicialmente é alocado dinamicamente um espaço de memória o qual será apontado pela variável aux. Caso esta alocação não seja possível, a variável aux continuará com o valor nil. Neste caso o procedimento escreve a mensagem Overflow na tela e finaliza o seu trabalho. Caso contrário, a alocação ocorreu, portanto, o espaço de memória está disponível para inserção do nó, então, o procedimento irá guardar no campo info do endereço apontado por aux o valor contido na variável D, guardar no campo prox do endereço apontado por aux o endereço apontado por P, ligando este novo nó a antiga estrutura, e guardar em P a

variável aux. Podemos dizer que a variável ponteiro P será o topo da nossa pilha, pois ela estará sempre apontando para o último nó inserido.

Esta é a operação de inserir um valor na pilha encadeada por alocação dinâmica de memória.

```
procedure desempilha (var P : apontno; var D : char);
var aux : apontno;
begin
  if (P = nil)
    then writeln ('Underflow')
    else begin
      D := P^.info;
      aux := P;
      P := P^.prox;
      dispose (aux);
    end;
end;
```

Programas 20 - Procedimento desempilha

O procedimento desempilha recebe o parâmetro P do tipo apontno e o parâmetro D do tipo char por referência. É declarado no procedimento uma variável auxiliar do tipo apontno que será utilizada na manipulação dos nós da estrutura.

Inicialmente é verificado se a variável P está vazia, caso esteja, o procedimento escreve a mensagem Underflow na tela e finaliza o seu trabalho. Caso contrário, o procedimento irá guardar na variável D o conteúdo do campo info do endereço apontado por P, para que esta variável retorne com o valor desempilhado. Posteriormente, a variável aux recebe P ficando estes dois ponteiros apontando para o mesmo nó, depois, a variável P passa a apontar para o endereço apontando pelo campo prox dessa mesma variável, e finalmente é realizado um dispose no endereço, removendo este nó da pilha obedecendo sua disciplina de acesso, além de devolver este endereço ao sistema operacional para ser utilizado em outro programa.

Esta é a operação de remover um valor na pilha encadeada por alocação dinâmica de memória.

2.4. Fila usando alocação dinâmica de memória

Assim como a Pilha, a Fila também pode ser implementada usando basicamente os recursos de encadeamento e alocação dinâmica de memória. Neste caso, sua estrutura ficaria conforme Figura 11.

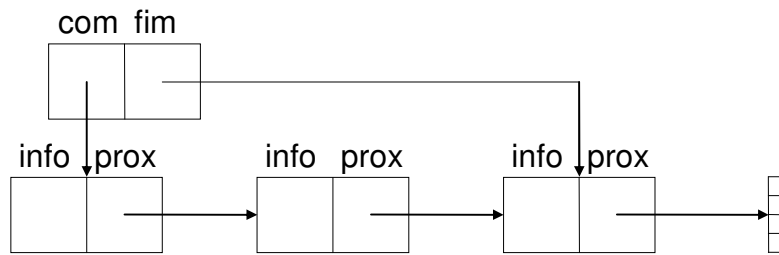


Figura 11 - Estrutura de fila encadeada por alocação dinâmica de memória

Vamos analisar como ficaria a implementação da fila utilizando alocação dinâmica de memória. Inicialmente iremos declarar um tipo capaz de receber os valores deste TAD.

```
type apontno = ^no;
  no = record
    info : string;
    lig : pontno;
  end;
  fila = record
    com : pontno;
    fim : pontno;
  end;
```

Programas 21 - Declaração do tipo fila encadeada

Para implementação da fila declaramos três tipos: apontno, no e fila. O tipo no é composto de dois campos:

- info: é do tipo char, nele será armazenado os valores da fila.
- lig: é do tipo apontno, ele irá indicar quem é o próximo no depois do atual.

O tipo apontno é do tipo endereço de no, ou seja, ele é um ponteiro capaz de guardar endereço de memória que tenha o formato do tipo no. Observe também que o campo prox do tipo no também é do tipo apontno, este fato justifica-se na utilização deste campo para realização do encadeamento proposto na criação da fila encadeada por alocação dinâmica de memória.

O tipo fila possui dois campos:

- com: é do tipo apontno, este campo irá apontar para o primeiro nó da fila.
- fim: é do tipo apontno, este campo irá apontar para o último nó da fila.

```
procedure iniciafila (var F : fila);  
begin  
    F.com := nil;  
    F.fim := nil;  
end;
```

Programas 22 - Procedimento iniciafila

O procedimento iniciafila recebe o parâmetro F por referência, que contém a nossa fila. A atribuição do valor nil para os campos com e fim indica que não temos nenhum valor na fila atualmente.

Esta é a operação de iniciar uma fila encadeada por alocação dinâmica de memória.

```
procedure inserefila (var F : fila; D : string);  
var aux : apontno;  
begin  
    new (aux);  
    if (aux = nil)  
        then writeln ('Fila cheia')  
    else begin  
        aux^.info := D;  
        aux^.lig := nil;  
        if F.fim <> nil  
            then F.fim^.lig := aux; Inserção em uma fila não vazia  
        F.fim := aux;  
        if F.com = nil  
            then F.com := aux; Inserção em uma fila vazia  
    end;
```

Programas 23 - Procedimento inserefila

O procedimento inserefila recebe o parâmetro F do tipo fila por referência e o parâmetro D do tipo string por valor. O parâmetro D deverá conter o valor a ser inserido na fila. É declarado no procedimento uma variável auxiliar do tipo apontno que será utilizada na criação da estrutura.

Inicialmente é alocado dinamicamente um espaço de memória o qual será apontado pela variável aux. Caso esta alocação não seja possível, a variável aux continuará com o valor nil. Neste caso o procedimento escreve a mensagem “Fila cheia” na tela e finaliza o seu trabalho. Caso contrário, a alocação ocorreu, portanto, o espaço de memória está disponível para inserção do nó, então, o procedimento irá

guardar no campo info do endereço apontado por aux o valor contido na variável D e guardar no campo lig do endereço apontado por aux o valor nil. Posteriormente, caso o fim da fila for diferente de nil indica que a fila não estava vazia, portanto, o campo lig apontado pelo campo fim da fila passa a apontar para o endereço do novo nó, o fim da fila passa a ser este novo nó obedecendo a disciplina de acesso da Fila onde a inserção de valores deve ocorrer no fim. Outro teste realizado será no caso de inserção em uma fila vazia, quando esta condição ocorrer, o campo com da fila será igual a nil, então, ele também deverá apontar para este novo nó que está apontado pela variável ponteiro aux.

Esta é a operação de inserir um valor na fila encadeada por alocação dinâmica de memória.

```
procedure retirafila (var F : fila; var D : string);  
var aux : apontno;  
begin  
  if F.com = nil  
    then writeln ('Fila vazia')  
    else begin  
      aux := F.com;  
      D := aux^.info;  
      F.com := aux^.lig;  
      if F.com = nil  
        then F.fim := nil;  
      dispose (aux);  
    end;  
end;
```

Retirar o último valor da fila

Programas 24 - Procedimento retirafila

O procedimento retira recebe o parâmetro F do tipo fila e o parâmetro D do tipo string por referência. É declarado no procedimento uma variável auxiliar do tipo apontno que será utilizada na manipulação dos nós da estrutura.

Inicialmente é verificado se o campo com da variável F está vazio, caso esteja, o procedimento escreve a mensagem “Fila vazia” na tela e finaliza o seu trabalho. Caso contrário, o procedimento irá apontar a variável aux para o começo da fila, depois guardar na variável D o conteúdo do campo info do endereço apontado por aux, para que esta variável retorne com o valor desempilhado obedecendo a disciplina de acesso da fila que institui a remoção no começo. Posteriormente, a o campo com da variável F recebe o endereço apontado pelo campo lig do endereço apontado por aux, caso com a remoção desse valor o começo passe a ser nil, atualiza-se o fim também para nil tornando a fila vazia e finalmente é realizado um dispose no endereço apontado por aux, o antigo começo, removendo este nó da fila.

Esta é a operação de remover um valor na fila encadeada por alocação dinâmica de memória.

2.5. Lista usando alocação dinâmica de memória

Conseguimos através do encadeamento resolver o problema relacionado com o baixo desempenho das primitivas de inserção e remoção de nó na lista por contigüidade devido ao grande número de operações necessárias para estes processos e com o recurso de alocação dinâmica de memória conseguimos resolver o problema da limitação do tamanho da lista.

Vamos analisar como ficaria a implementação da lista utilizando alocação dinâmica de memória. Inicialmente iremos declarar um tipo capaz de receber os valores deste TAD.

```
type lista = ^no;  
  no = record  
    info : string;  
    lig : lista;  
  end;
```

Programas 25 - Declaração do tipo lista por alocação dinâmica de memória

Para implementação da lista declaramos dois tipos: lista e no. O tipo no é composto de dois campos:

- info: é do tipo string, nele será armazenado os valores da lista.
- lig: é do tipo lista, ele irá indicar quem é o próximo no depois do atual.

O tipo lista é do tipo endereço de no, ou seja, ele é um ponteiro capaz de guardar endereço de memória que tenha o formato do tipo no. Observe também que o campo lig do tipo no também é do tipo lista, este fato justifica-se na utilização deste campo para realização do encadeamento proposto na criação da lista encadeada por alocação dinâmica de memória.

```
procedure inicialista (var L:lista);  
begin  
  L := nil;  
end;
```

Programas 26 - Procedimento inicialista

O procedimento inicialista recebe o parâmetro L por referência, que contém a nossa lista. A atribuição do valor nil para esta variável indica que não temos nenhum valor na lista atualmente.

Esta é a operação de iniciar uma lista encadeada por alocação dinâmica de memória.

```
procedure inserelista (var L:lista; D:string);
var aux, ant : lista;
begin
  aux := L;
  ant := nil;
  while (aux <> nil) and (aux^.info<=D) do
  begin
    ant := aux;
    aux := aux^.lig;
  end;
  new (aux);
  aux^.info := D;
  if ant = nil then
  begin
    aux^.lig := L;
    L := aux;
  end
  else begin
    aux^.lig := ant^.lig;
    ant^.lig := aux;
  end;
end;
```

Loop que encontra a posição de inserção do valor na lista

Inserção no começo

Inserção em posições posteriores a primeira

Programas 27 - Procedimento inserelista

O procedimento inserelista recebe o parâmetro L do tipo lista por referência e o parâmetro D do tipo string por valor. O parâmetro D deverá conter o valor a ser inserido na lista. É declarado no procedimento as variáveis **aux** e **ant** as quais são auxiliares no posicionamento dos valores.

Iniciando, a variável aux aponta para a lista e a variável ant recebe nil. Posteriormente, elas entram em um loop onde o objetivo é encontrar a posição onde o valor a ser inserido deverá entrar posicionando a variável aux na posição posterior a do valor que será inserido e a variável ant na posição anterior deste mesmo valor, para isso, o loop acontece enquanto aux for diferente de nil indicando que chegou ao fim e o valor da lista, o qual satisfeito as condições irá passar por todos, for menor ou igual ao valor a inserir.

Findado o loop, é realizado uma alocação dinâmica de memória e o endereço alocado é guardado na variável ponteiro aux. Depois é guardado no campo info do endereço apontado pela variável aux o valor a ser inserido. Inserido o valor, o encadeamento da lista segue dois critérios: inserção no começo, onde o valor da variável ponteiro ant for nil, neste caso o lig do no inserido irá receber o no apontado por L e ele passará a ser o novo começo da lista; inserção nas próximas posições onde o campo lig do no inserido recebe o lig da posição anterior e o lig da anterior recebe o endereço do no inserido.

Esta é a operação de inserir um valor na lista encadeada por alocação dinâmica de memória.

```
procedure removelista (var L:lista; n:string);  
var aux, ant: lista;  
begin  
  if L = nil  
  then write ('Lista vazia')  
  else begin  
    aux := L;  
    ant := nil;  
    while (aux^.info<>n) and (aux<>nil) do  
    begin  
      ant := aux;  
      aux := aux^.lig;  
    end;  
    if aux = nil  
    then write ('Nome nao encontrado')  
    else begin  
      if aux = L  
      then L := L^.lig  
      else ant^.lig := aux^.lig;  
      dispose (aux);  
    end;  
  end;  
end;
```

Loop que encontra a posição de remoção do valor na lista

Programas 28 - Procedimento removelista

O procedimento removelista recebe o parâmetro L por referência, que contém a nossa lista e a variável n que terá o valor a ser removido. No procedimento são declaradas as variáveis **aux** e **ant** as quais são auxiliares no posicionamento dos valores.

Inicialmente é verificado se existe valores na lista, testando se a variável L = nil, caso seja, o procedimento escreve a mensagem “Lista vazia” e encerra o seu

trabalho. Caso contrário, a variável aux aponta para a lista e a variável ant recebe nil. Posteriormente, elas entram em um loop onde o objetivo é posicionar a variável aux na posição do valor que será removido e a variável ant na posição anterior deste mesmo valor, para isso, o loop acontece enquanto o valor da lista, o qual satisfeito as condições irá passar por todos, for diferente do valor a ser removido e aux diferente de nil indicando que chegou ao fim.

Findado o loop, é realizado um teste na variável aux, caso ela seja igual a nil indica que o valor a ser removido não está na lista, portanto o procedimento escreve a mensagem “Nome não encontrado” e finaliza seu trabalho. Caso contrário, o valor deverá ser removido, porém a atualização do encadeamento segue dois critérios: remoção do primeiro nó, neste caso o começo deverá ser o campo lig do antigo começo apontado pela variável L; remoção de valores posteriores, o lig do endereço apontado pela variável ant recebe o lig do endereço apontado pelo nó o qual deverá ser removido. Posteriormente, o nó é removido da lista utilizando o comando dispose devolvendo o endereço alocado dinamicamente para o sistema operacional.

Esta é a operação de remover um valor na lista encadeada por alocação dinâmica de memória.

REFERÊNCIAS

(DROZDEK, 2002) DROZDEK, Adam. **Estrutura de Dados e Algoritmos em C++**. São Paulo: Pioneira Thompson Learning, 2002.

PEREIRA, S. **Estrutura de Dados Fundamentais: Conceitos e Aplicações**. São Paulo: Érica, 1996.

(PIMENTEL, 2007) PIMENTEL, Graça. Material didático da disciplina Algoritmos e Estruturas de Dados. Disponível em <http://www.icmc.usp.br/~sce182/> acesso em 11/2007.

(PREISS, 2000) PREISS, Bruno R. **Estruturas de Dados e Algoritmos: padrões de projetos orientados a objetos com Java**. Campus, 2000.

VELOSO, Paulo et al. **Estruturas de Dados**. Rio de Janeiro: Campus, 1983. 228 p.

(ZIVIANI, 2005) ZIVIANI, Nívio. **Projeto de Algoritmos: com implementações em Pascal e C**. 2 ed. ver. e ampl. - São Paulo: Pioneira Thompson Learning, 2005.