Estruturas de Dados

Prof. Anderson Grandi Pires (CEFET-MG)

Observação: Este documento não tem a intenção de explorar todas as opções e funcionalidades para criação de pseudocódigos, mas sim estabelecer um padrão a ser utilizado nesta disciplina. Além disso, a preocupação maior aqui é apresentar as estruturas de dados de forma didática, sem a preocupação de abordar todos os seus aspectos ou um maior aprofundamento. Isso será feito nas aulas e nas atividades de laboratório, quando tais conceitos serão implementados em uma linguagem de programação.

SUMÁRIO

Observação: Este material está em constante atualização. Caso seja observada alguma inconsistência ou erro, favor enviar mensagem para o professor (agpires@cefetmg.br).

Notação para pseudocódigo

Simbologia básica

operador de atribuição	←
operadores relacionais	**= ≠ > < ⩾ ⩽**
operadores lógicos	**e ou não**
operadores aritméticos	**+ - / ***
operador de indexação de arranjos	**[]**
alguns tipos de dados	**lógico, inteiro, real, caractere, literal**
operador piso	⌊.⌋ Ex: ⌊**x**⌋ é o maior inteiro menor ou igual a **x**
operador teto	⌈.⌉ Ex: ⌈**x**⌉ é o menor inteiro maior ou igual a **x**
expressão lógica	formada de operadores **lógicos** ou **relacionais** (resulta em **verdadeiro** ou **falso**)
expressão aritmética	formada por operadores **aritméticos** (resulta em um número)

Precedência de operadores

Indica que operadores devem ser avaliados primeiro. Os primeiros operadores têm mais precedência que os últimos, ou seja, devem ser avaliados primeiro.

```
( )  []  ⌊.⌋  ⌈.⌉
não
* /
+ -
> < ⩾ ⩽
= ≠
e
ou
←
```

Exemplos

```
Atribuição e acesso a valores:
x ← 1            atribuição a x
y ← x + 5        acesso ao conteúdo de x e atribuição a y

Expressão lógica:
resultado ← y ⩾ 10 e 6 ≠ x
Teste ← y = 6 e y ≠ x

Incremento e decremento:
i ← 0
i ← i + 1
j ← j - 2

Acesso a arranjos (vetores):
k ← vetor[0]
vet[i] ← x*2
vet[i+1] ← vet[i]
vet[i] ← vet[i-1]

Operadores piso e teto:
X ← 3 + ⌈7/2⌉            resultado desta operação: X ← 3 + 4
Y ← 2 * ⌊7/2⌋            resultado desta operação: Y ← 2 * 3

Acesso a registros (notação ponto):
pessoa.idade             pessoa é um registro que possui um
                         campo/membro idade
```

Estrutura de controle sequencial

A análise ou leitura de um pseudocódigo se dá de **cima para baixo** e da **esquerda para a direita**. A numeração abaixo à esquerda indica a ordem da leitura. Em caso de **atribuição**, a expressão do lado direito da instrução deverá ser calculada antes da atribuição ser efetuada. Isto é uma consequência da aplicação da **ordem de precedência** dos operadores.

```
i ← 0
vetor ← {0, 1, 2, 3, 4, 5}
i ← i + 1
j ← j - 2
resultado ← y ⩾ 10 e 6 ≠ x
k ← vetor[0]
j ← k + vetor[i]
resultado ← verdadeiro ou i > k
```

Estruturas de controle condicional

Condicional Simples

```
se condição então
  instrução_1
  …
  instrução_n
```

Condicional Composta

```
se condição então
  instruções associadas à cláusula então
senão
  instruções associadas à cláusula senão
```

Observação: **condição** é uma expressão lógica

Exemplos

```
se a > b então
  x ← x + 1

se x > y então
  resultado ← x
senão
  resultado ← y

se x ⩾ 6 então
  resultado ← "aprovado"
senão se x ⩾ 4 então
  resultado ← "exame especial"
senão
  resultado ← "reprovado"
```

Estruturas de controle do tipo repetição

Variável de controle

```
para i ← 1 até k passo p faça
  instrução_1
  …
  instrução_n
```

Observações:

⇒ i ← 1 **até** k **passo** p indica que o valor de **i** *inicia* em **1** sendo *incrementado* em **p** unidades *até superar* **k**, momento em que a estrutura de repetição finaliza

⇒ quando o valor do passo for 1, pode deixar este informação implícita, como segue: **para** i ← 1 **até** k **faça**

Exemplos

```
k ← 10
v ← 1
para i ← 1 até k passo 1 faça
  v ← v + i
para i ← 1 até k faça
  v ← v + i
v ← 0
para i ← 1 até k passo 2 faça
  i ← i +1
  v ← v + i
```

Teste no início

```
enquanto condição faça
  instrução_1
  …
  instrução_n
```

Observação: condição é uma expressão lógica

Exemplo

```
k ← 15
contador ← 0
enquanto k > 0 faça
  k ← k - 1
  contador ← contador + 1
```

Teste no fim

```
faça
  instrução_1
  …
  instrução_n
enquanto condição
```

Exemplo

```
i ← 0
faça
  vet[i] ← vet[i] * 2
  i ← i + 1
enquanto i ⩽ 10
```

Sub-rotinas

```
tipo_retorno id_subrotina(lista_parametros)          cabeçalho
Entrada: descrição dos parâmetros
Saída: descrição do valor de retorno
var_aux: descrição do tipo e propósito da variável var_aux

  instrução_1
  ...
  instrução_n
```

Observações:

(i) Caso **não** haja um retorno associado à sub-rotina, ela não terá um tipo de retorno

(ii) Caso haja mais de uma variável auxiliar, utilizar uma linha para descrever cada uma

Exemplos

```
inteiro somar(v1, v2)
Entrada: valores inteiros a serem somados
Saída: resultado da soma de v1 e v2
soma: variável auxiliar para o cálculo

  soma ← v1 + v2

  retorna soma
```

```
real somar(v, tam)
Entrada: vetor v com tam elementos reais
Saída: resultado da soma dos elementos presentes no vetor
soma: variável auxiliar para o cálculo

  soma ← 0
  para i ← 1 até tam faça
    soma ← soma + v[i]

  retorna soma
```

```
incrementar(v, tam, inc)
Entrada: vetor v com tam elementos reais e a quantidade inc a ser
  incrementada
Saída: Não possui

  para i ← 1 até tam faça
    v[i] ← v[i] + inc
```

Observação: *A sub-rotina incrementar não possui tipo de retorno*

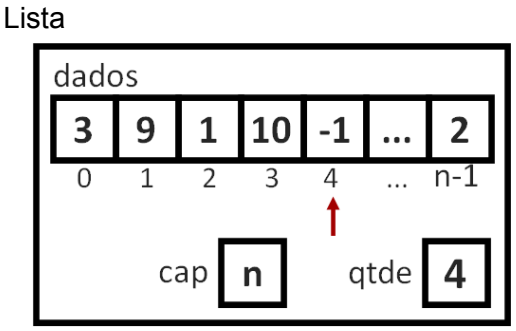
Algumas sub-rotinas úteis

```
inteiro quociente(x, y)
Entrada: números inteiros x e y
Saída: resultado da divisão inteira de x por y
      se y ≠ 0 então
            retorna ⌊x/y⌋
      senão
            retorna x
```

```
inteiro resto(x, y)
Entrada: números inteiros x e y
Saída: resto da divisão inteira de x por y
      se y ≠ 0 então
            retorna x - ⌊x/y⌋ * y
      senão
            retorna x
```

Listas Lineares Sequenciais

Representação

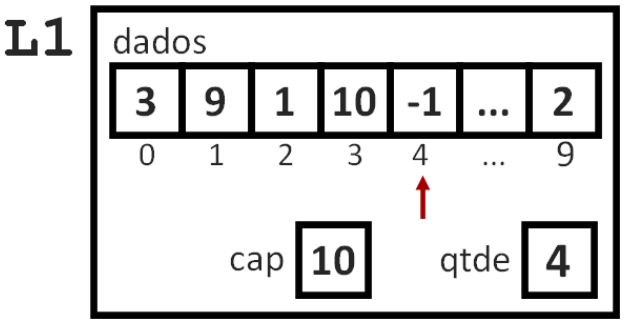


Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento/item estará na posição de índice `0` e o último na posição de índice `qtde-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `qtde`, o qual indica a próxima posição **livre**. Em uma lista com pelo menos 1 item, os itens válidos da lista são aqueles posicionados desde o índice `0` até o índice `qtde-1`.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da lista (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.)
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos/itens que a lista poderá armazenar.
- A representação acima compreende somente a parte de dados do TAD *Lista Linear Sequencial*, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Acesso a dados

Seja uma lista L1, conforme figura abaixo:



Operações de acesso aos dados da lista

`L1.dados`: acesso ao campo/membro `dados` da lista `L1` (`dados` é um arranjo/vetor)
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L1.cap`: acesso ao campo/membro `cap` da lista `L1` (`cap` armazena o número máximo de itens que se pode armazenar na lista, ou seja, sua capacidade)
`L1.dados[i]`: acesso ao item situado na posição de índice `i` do campo `dados` da lista `L1`

Operações auxiliares

Capacidade da lista

```
inteiro capacidade(L)
Entrada: lista L
Saída: capacidade da lista

   retorna L.cap
```

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens válidos
       presentes na lista

   retorna L.qtde
```

Verificação de lista cheia

```
lógico cheia(L)
Entrada: lista L
Saída: verdadeiro se estiver cheia; falso caso contrário

   se L.qtde = L.cap então
      retorna verdadeiro

   retorna falso
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operações de inserção

Inserção no início

```
lógico inserir_inicio(L, e)
Entrada: lista L, elemento e a ser inserido
Saída: sucesso ou falha na operação

   se cheia(L) então
      retorna falso

   para i ← L.qtde até 1 passo -1 faça
      L.dados[i] ← L.dados[i-1]

   L.dados[0] ← e
   L.qtde ← L.qtde + 1

   retorna verdadeiro
```

Inserção no meio

```
lógico inserir_meio(L, e, k)
Entrada: lista L, elemento e a ser inserido, k-ésima posição onde o
  elemento e será inserido
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se cheia(L) então
      retorna falso

   para i ← L.qtde até k passo -1 faça
      L.dados[i] ← L.dados[i-1]

   L.dados[k-1] ← e
   L.qtde ← L.qtde + 1

   retorna verdadeiro
```

Inserção no fim

```
lógico inserir_fim(L, e)
Entrada: lista L, elemento e a ser inserido
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se cheia(L) então
      retorna falso

   L.dados[L.qtde] ← e
   L.qtde ← L.qtde + 1

   retorna verdadeiro
```

Operações de remoção

Remoção no início

```
lógico remover_inicio(L)
Entrada: lista L
Saída: sucesso ou falha na operação

   se vazia(L) então
      retorna falso

   para i ← 1 até tamanho(L)-1 faça
      L.dados[i-1] ← L.dados[i]

   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

Remoção no meio

```
lógico remover_meio(L, k)
Entrada: lista L, k-ésima posição na qual um elemento será removido
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se vazia(L) ou invalido(k) então
      retorna falso

   para i ← k-1 até tamanho(L)-1 faça
      L.dados[i] ← L.dados[i+1]

   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

Remoção no fim

```
lógico remover_fim(L)
Entrada: lista L
Saída: sucesso ou falha na operação

   se vazia(L) então
      retorna falso

   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

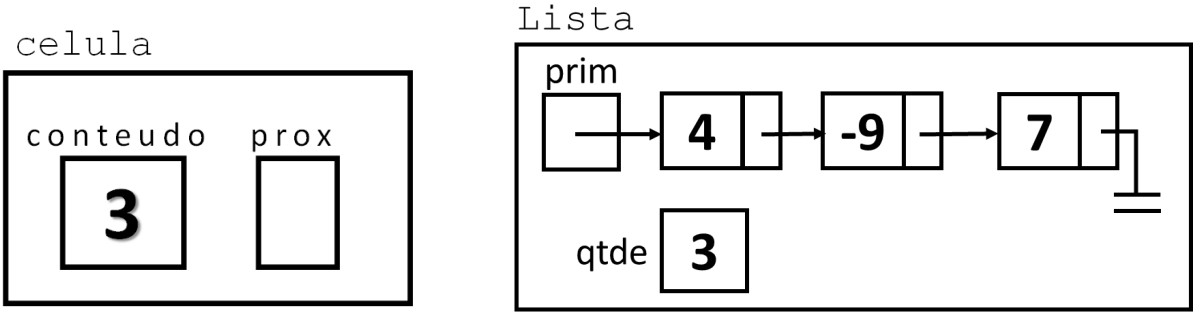
Operação de busca

Busca sequencial

```
lógico buscar_seq(L, e)
Entrada: lista L, elemento e a ser procurado na lista
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se vazia(L) então
      retorna falso

   para i ← 0 até tamanho(L)-1 faça
      se L.dados[i] = e então
         retorna verdadeiro

   retorna falso
```

Listas Lineares Encadeadas

Representação

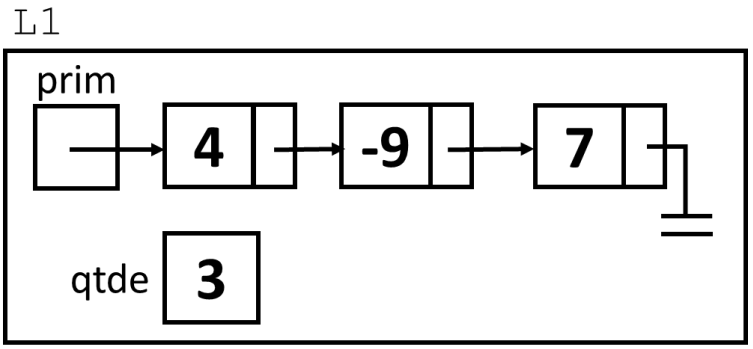


Algumas regras

- Os itens presentes nas células da lista são acessados por meio do campo (membro) `prim`, sendo esse uma referência para o primeiro item da lista. Caso a referência seja nula, a lista estará vazia. O campo (membro) `qtde` indica o número de itens presentes na lista.
- Os itens na lista não estão indexados, porém mantém uma ordem: 1ª item da lista, 2ª item da lista, etc. O acesso à primeira célula da lista é feito por meio do campo `prim`, enquanto que o acesso aos demais itens é feito utilizando as referências armazenadas no campo `prox` de cada célula.
- A representação acima compreende somente a parte de dados do TAD Lista Linear Encadeada, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Acesso a dados

Seja uma lista L1, conforme figura abaixo:



Operações de acesso aos dados da lista:

`L1.prim`: acesso ao campo/membro `prim` da lista `L1`
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens presentes na lista)
`L1.prim.prox`: acesso ao campo/membro `prox` da primeira célula da lista `L1`
`L1.prim.conteudo`: acesso ao campo/membro `conteudo` da primeira célula da lista `L1`

Observações

Assume-se que os argumentos passados para as chamadas às sub-rotinas estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.

Operações auxiliares

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens presentes na
       lista

   retorna L.qtde
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Observação:

As sub-rotinas `capacidade` e `cheia` não fazem sentido neste tipo de lista, portanto não sendo apresentadas

Operações de inserção

Inserção no início

```
lógico inserir_inicio(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: sucesso ou falha na operação
cel: célula (nó) a ser inserida

  se invalido(ctd) então
     retorna falso

  cel.conteudo ← ctd
  cel.prox ← L.prim
  L.prim ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Inserção no meio

```
lógico inserir_meio(L, ctd, k)
Entrada: lista L, conteúdo ctd, k-ésima posição
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) ou invalido(k) então
    retorna falso

  cel.conteudo ← ctd
  se k = 1 então
    cel.prox ← L.prim
    L.prim ← cel
  senão
    temp ← L.prim
    para i ← 1 até k-2 passo 1 faça
      temp ← temp.prox
    cel.prox ← temp.prox
    temp.prox ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Inserção no fim

```
lógico inserir_fim(L, ctd)
Entrada: lista L, conteúdo ctd a inserir
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) então
    retorna falso

  cel.conteudo ← ctd
  cel.prox ← NULO

  se esta_vazia(L) então
    L.prim ← cel
  senão
    temp ← L.prim
    enquanto temp.prox ≠ NULO faça
      temp ← temp.prox
    temp.prox ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Operações de remoção

Remoção no início

```
lógico remover_inicio(L)
Entrada: lista L
Saída: sucesso ou falha na operação

   se vazia(L) então
      retorna falso

   L.prim ← L.prim.prox
   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

Remoção no meio

```
lógico remover_meio(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação

   se vazia(L) ou invalido(k) então
      retorna falso

   temp ← L.prim
   se L.qtde = 1 então
      L.prim ← NULO
   senão
      para i ← 1 até k-2 passo 1 faça
         temp ← temp.prox
      temp.prox ← temp.prox.prox

   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

Remoção no fim

```
lógico remover_fim(L)
Entrada: lista L
Saída: sucesso ou falha na operação

   se vazia(L) então
      retorna falso

   se L.qtde = 1 então
      L.prim ← NULO
   senão
      temp ← L.prim
      enquanto temp.prox.prox ≠ NULO faça
         temp ← temp.prox
      temp.prox ← NULO

   L.qtde ← L.qtde - 1

   retorna verdadeiro
```

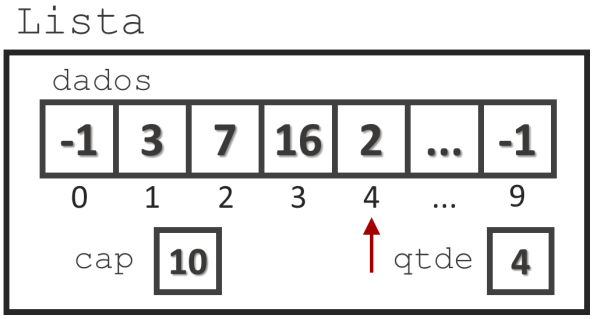
Operação de busca

Busca sequencial

```
lógico buscar(L, ctd)
Entrada: lista L, conteúdo ctd a ser buscado
Saída: sucesso ou falha na operação

   se vazia(L) então
      retorna falso

   temp ← L.prim
   enquanto temp ≠ NULO faça
      se temp.conteudo = ctd então
            retorna verdadeiro
      temp ← temp.prox

   retorna falso
```

Listas Lineares Sequenciais Ordenadas

Representação

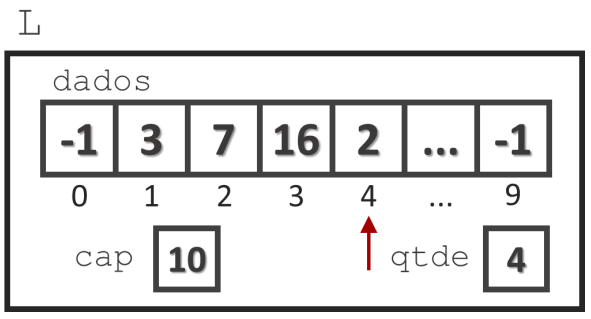


Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento/item estará na posição de índice `0` e o último na posição de índice `qtde-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `qtde`, o qual indica a próxima posição **livre**. Em uma lista com pelo menos 1 item, os itens válidos da lista são aqueles posicionados desde o índice `0` até o índice `qtde-1`.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da lista (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.)
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos/itens que a lista poderá armazenar.
- A representação acima compreende somente a parte de dados do TAD *Lista Linear Sequencial*, sendo que a forma de acesso e as principais operações serão definidas em seguida.
- **A lista deve permanecer ordenada após operações de inserção ou remoção.**

Acesso a dados

Seja uma lista L, conforme figura abaixo:



Operações de acesso aos dados da lista

`L.dados`: acesso ao campo/membro `dados` da lista `L` (`dados` é um arranjo/vetor)
`L.qtde`: acesso ao campo/membro `qtde` da lista `L` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L.cap`: acesso ao campo/membro `cap` da lista `L` (`cap` armazena o número máximo de itens que se pode armazenar na lista, ou seja, sua capacidade)
`L.dados[i]`: acesso ao item situado na posição de índice `i` do campo `dados` da lista `L`

Operações auxiliares

Capacidade da lista

```
inteiro capacidade(L)
Entrada: lista L
Saída: capacidade da lista

   retorna L.cap
```

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens válidos
       presentes na lista

   retorna L.qtde
```

Verificação de lista cheia

```
lógico cheia(L)
Entrada: lista L
Saída: verdadeiro se estiver cheia; falso caso contrário

   se L.qtde = L.cap então
      retorna verdadeiro

   retorna falso
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

```
lógico inserir(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: verdadeiro se inserido; falso caso contrário
  se cheia(L) ou invalido(ctd) então
      retorna falso
  se vazia(L) então
      L.dados[0] ← ctd
  senão
      i ← L.qtde
      enquanto i > 0 e L.dados[i-1] > ctd faça
            L.dados[i] ← L.dados[i-1]
            i ← i - 1
      L.dados[i] ← ctd
  L.qtde ← L.qtde + 1
  retorna verdadeiro
```

Operação de remoção

Remoção de item na k-ésima posição

```
lógico remover(L, k)
Entrada: lista L, k-ésima posição
Saída: verdadeiro se inserido; falso caso contrário
  se vazia(L) ou invalido(k) então
      retorna falso
  para i ← k-1 até tamanho(L)-2 faça
      L.dados[i] ← L.dados[i+1]
  L.qtde ← L.qtde - 1
  retorna verdadeiro
```

Remoção de item com chave ctd

```
lógico remover(L, c)
Entrada: lista L, item com chave c a ser removido
Saída: sucesso (verdadeiro) ou falha (falso) na operação
  se vazia(L) então
      retorna falso
  para i ← 0 até tamanho(L)-1 faça
      se L.dados[i] = ctd então
            para j ← i até tamanho(L)-2 faça
                  L.dados[j] ← L.dados[j+1]
            L.qtde ← L.qtde - 1
            retorna verdadeiro
  retorna falso
```

Operações de busca

Busca sequencial

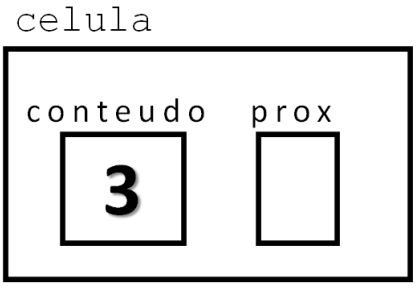
```
lógico buscar_seq(L, e)
Entrada: lista L, elemento e a ser procurado na lista
Saída: verdadeiro se elemento encontrado; falso caso contrário
   se vazia(L) então
      retorna falso
   para i ← 0 até tamanho(L)-1 faça
      se L.dados[i] = e então
         retorna verdadeiro
   retorna falso
```

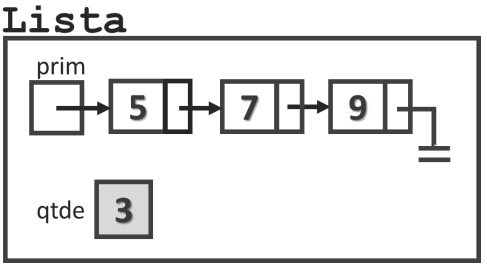
Busca binária

```
lógico busca_binaria(L, valor)
Entrada: lista L
Saída: verdadeiro se valor encontrado; falso caso contrário
  se vazia(L) então
      retorna falso
  esq ← 0
  dir ← tamanho(L) - 1
  enquanto esq ≤ dir faça
      meio ←(esq + dir)/2
      se L.dados[meio] = valor então
         retorna verdadeiro
      senão
         se valor > L.dados[meio] então
           esq ← meio + 1
         senão
           dir ← meio - 1

  retorna falso
```

Listas Lineares Encadeadas Ordenadas

Representação



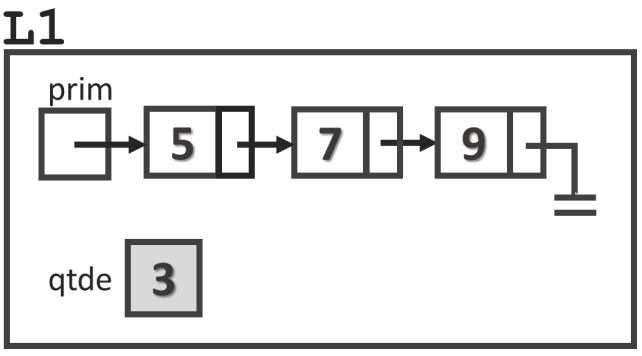


Algumas regras

- Os itens presentes nas células da lista são acessados por meio do campo (membro) `prim`, sendo esse uma referência para o primeiro item da lista. Caso a referência seja nula, a lista estará vazia. O campo (membro) `qtde` indica o número de itens presentes na lista.
- Os itens na lista não estão indexados, porém mantém uma ordem: 1ª item da lista, 2ª item da lista, etc. O acesso à primeira célula da lista é feito por meio do campo `prim`, enquanto que o acesso aos demais itens é feito utilizando as referências armazenadas no campo `prox` de cada célula.
- A representação acima compreende somente a parte de dados do TAD Lista Linear Encadeada, sendo que a forma de acesso e as principais operações serão definidas em seguida.
- **A lista deve permanecer ordenada após operações de inserção ou remoção.**

Acesso a dados

Seja uma lista L1, conforme figura abaixo:



Operações de acesso aos dados da lista:

`L1.prim`: acesso ao campo/membro `prim` da lista `L1`
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens presentes na lista)
`L1.prim.prox`: acesso ao campo/membro `prox` da primeira célula da lista `L1`
`L1.prim.conteudo`: acesso ao campo/membro `conteudo` da primeira célula da lista `L1`

Observações

Assume-se que os argumentos passados para as chamadas às sub-rotinas estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.

Operações auxiliares

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens presentes na
       lista

   retorna L.qtde
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Observação:

As sub-rotinas `capacidade` e `cheia` não fazem sentido neste tipo de lista, portanto não sendo apresentadas.

Operações de inserção

```
lógico inserir(L, ctd)
Entrada: lista L, conteúdo ctd
Saída: sucesso ou falha na operação
  se invalido(ctd) então
      retorna falso
  cel.conteudo ← ctd
  temp ← L.prim
  se vazia(L) ou temp.conteudo > ctd então
      cel.prox ← L.prim
      L.prim ← cel
      L.qtde ← L.qtde + 1
      retorna verdadeiro
  senão
      para i ← 1 até tamanho(L)-1 faça
            se temp.prox.conteudo > ctd então
                  cel.prox ← temp.prox
                  temp.prox ← cel
                  L.qtde ← L.qtde + 1
                  retorna verdadeiro
            temp ← temp.prox
  retorna falso
```

Operações de remoção

Remoção de item na k-ésima posição

```
lógico remover(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso (verdadeiro) ou falha (falso) na operação
  se vazia(L) ou invalido(k) então
      retorna falso
  temp ← L.prim
  se k = 1 então
      L.prim ← L.prim.prox
  senão
      para i ← 1 até k-2 faça
            temp ← temp.prox
      temp.prox ← temp.prox.prox
  L.qtde ← L.qtde - 1
  retorna verdadeiro
```

Remoção de item com chave ctd

```
lógico remover(L, ctd)
Entrada: lista L, ctd a ser removido
Saída: sucesso (verdadeiro) ou falha (falso) na operação
  se vazia(L) então
      retorna falso

  temp ← L.prim
  se temp ≠ NULO e temp.conteudo = ctd então
      L.prim ← L.prim.prox
      L.qtde ← L.qtde - 1
      retorna verdadeiro
  senão
      enquanto temp.prox ≠ NULO faça
            se temp.prox.conteudo = ctd então
                  temp.prox ← temp.prox.prox
                  L.qtde  L.qtde - 1
                  retorna verdadeiro
            temp ← temp.prox
  retorna falso
```

Operação de busca

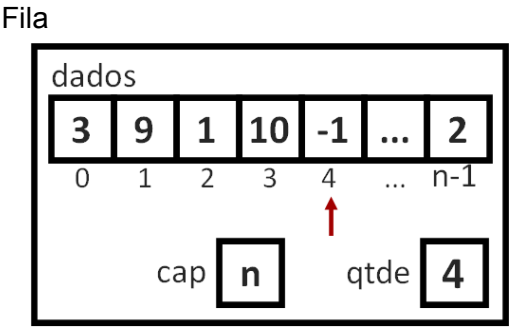
Busca sequencial

```
lógico buscar(L, ctd)
Entrada: lista L, conteúdo ctd a ser buscado
Saída: sucesso ou falha na operação
   se vazia(L) então
      retorna falso
   temp ← L.prim
   enquanto temp ≠ NULO faça
      se temp.conteudo = ctd então
            retorna verdadeiro
      temp ← temp.prox
   retorna falso
```

Filas (sequencial)

As filas nada mais são do que listas com restrição de acesso e manipulação. Desse modo, elas podem ser implementadas usando listas sequenciais ou listas encadeadas. Para tanto, a seguir são apresentados os dois tipos de listas e as respectivas alterações para a representação de filas.

Representação com listas sequenciais



Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento/item estará na posição de índice `0` e o último na posição de índice `qtde-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `qtde`, o qual indica a próxima posição **livre**. Em uma lista com pelo menos 1 item, os itens válidos da lista são aqueles posicionados desde o índice `0` até o índice `qtde-1`.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da fila ou pilha (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.).
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos/itens que a lista poderá armazenar.
- A representação acima compreende somente a parte de dados do TAD *Fila* ou TAD *Pilha*, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Operação auxiliar

Verificação de fila vazia

```
lógico vazia(F)
Entrada: fila F
Saída: verdadeiro se estiver vazia; falso caso contrário

   se F.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Inserção no fim (é o tipo de inserção usado em filas). Operação comumente denominada **push**.

```
lógico inserir(F, e)
Entrada: fila F, elemento e a ser inserido
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se cheia(F) então
      retorna falso

   F.dados[F.qtde] ← e
   F.qtde ← F.qtde + 1

   retorna verdadeiro
```

Operação de remoção

Remoção no início (é o tipo de inserção usado em filas). Comumente denominada **pop**.

```
lógico remover(F)
Entrada: fila F
Saída: sucesso ou falha na operação

   se vazia(F) então
      retorna falso

   para i ← 1 até tamanho(F)-1 faça
      F.dados[i-1] ← F.dados[i]

   F.qtde ← F.qtde - 1

   retorna verdadeiro
```

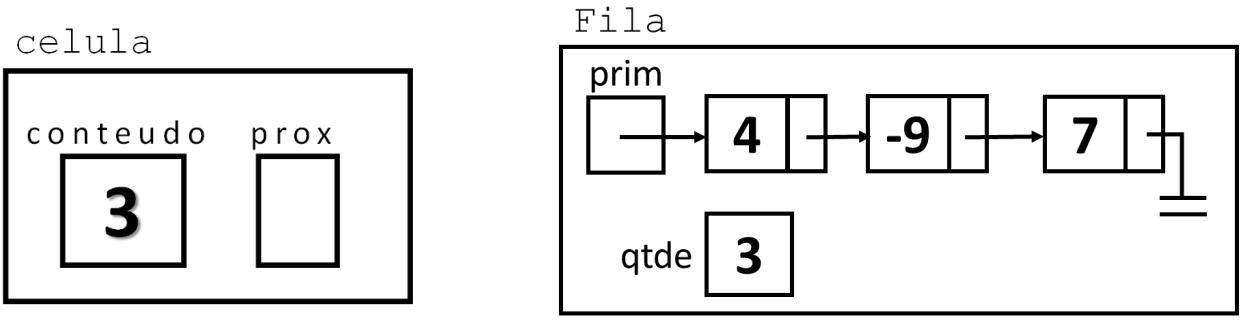
Operação de acesso

Em uma fila **somente** temos acesso ao elemento que está no início da fila (frente da fila). Esta operação assume que a fila não está vazia. O usuário dessa operação deverá testar a fila antes de acessar. O tipo `elemento` indica um tipo genérico para informar o retorno de um elemento presente na frente da fila. Para cada fila, `elemento` será compatível com o tipo de dado armazenado. Esta operação somente deve ser chamada quando a fila não estiver vazia. Comumente denominada de **front**.

```
elemento frente(F)
Entrada: fila F
Saída: sucesso ou falha na operação

   retorna F.dados[0]
```

Filas (representação com listas encadeadas)

Representação com listas encadeadas



Observações

Assume-se que os argumentos passados para as chamadas às sub-rotinas estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.

Operação auxiliar

Verificação de fila vazia

```
lógico vazia(F)
Entrada: fila F
Saída: verdadeiro se estiver vazia; falso caso contrário

   se F.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Na fila usamos a **inserção no fim**, logo o pseudocódigo abaixo é o mesmo desta funcionalidade. Operação comumente denominada **push**.

```
lógico inserir(F, ctd)
Entrada: fila F, conteúdo ctd a inserir
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) então
    retorna falso

  cel.conteudo ← ctd
  cel.prox ← NULO

  se vazia(L) então
    F.prim ← cel
  senão
    temp ← F.prim
    enquanto temp.prox ≠ NULO faça
      temp ← temp.prox
    temp.prox ← cel

  F.qtde ← F.qtde + 1

  retorna verdadeiro
```

Operação de remoção

Na fila usamos a **remoção no início**, logo o pseudocódigo abaixo é o mesmo desta funcionalidade. Comumente denominada **pop**.

```
lógico remover(F)
Entrada: fila F
Saída: sucesso ou falha na operação

   se vazia(F) então
      retorna falso

   F.prim ← F.prim.prox
   F.qtde ← F.qtde - 1

   retorna verdadeiro
```

Operação de acesso

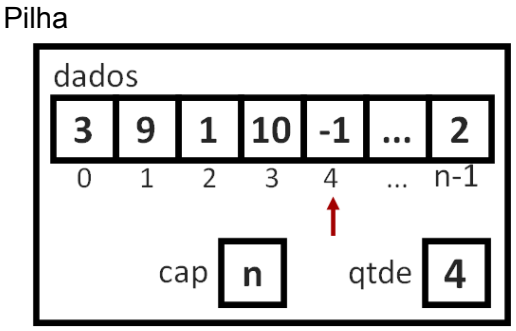
Em uma fila **somente** temos acesso ao elemento que está no início da fila (frente da fila). Esta operação assume que a fila não está vazia. O usuário dessa operação deverá testar a fila antes de acessar. O tipo `elemento` indica um tipo genérico para informar o retorno de um elemento presente na frente da fila. Para cada fila, `elemento` será compatível com o tipo de dado armazenado. Esta operação somente deve ser chamada quando a fila não estiver vazia. Comumente denominada de **front**.

```
elemento frente(F)
Entrada: fila F
Saída: sucesso ou falha na operação

   retorna F.prim.conteudo
```

Pilha (sequencial)

As pilhas nada mais são do que listas com restrição de acesso e manipulação. Desse modo, elas podem ser implementadas usando listas sequenciais ou listas encadeadas. Para tanto, a seguir são apresentados os dois tipos de listas e as respectivas alterações para a representação de pilhas.

Representação com listas sequenciais



Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento/item estará na posição de índice `0` e o último na posição de índice `qtde-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `qtde`, o qual indica a próxima posição **livre**. Em uma lista com pelo menos 1 item, os itens válidos da lista são aqueles posicionados desde o índice `0` até o índice `qtde-1`.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da fila ou pilha (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.).
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos/itens que a lista poderá armazenar.
- A representação acima compreende somente a parte de dados do TAD *Pilha*, sendo que a forma de acesso e as principais operações serão definidas em seguida.
- A inserção e remoção na pilha devem ocorrer na mesma extremidade. Por ter uma estrutura sequencial, neste TAD faremos a inserção e remoção no fim. Esta estratégia evita movimentação de dados.

Operação auxiliar

Verificação de pilha vazia

```
lógico vazia(P)
Entrada: pilha P
Saída: verdadeiro se estiver vazia; falso caso contrário

   se P.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Inserção no fim (é o tipo de inserção usado em pilhas). Operação comumente denominada **push**.

```
lógico inserir(P, e)
Entrada: pilha P, elemento e a ser inserido
Saída: sucesso (verdadeiro) ou falha (falso) na operação

   se cheia(P) então
      retorna falso

   P.dados[P.qtde] ← e
   P.qtde ← P.qtde + 1

   retorna verdadeiro
```

Operação de remoção

Remoção no fim (é o tipo de inserção usado em pilha). Comumente denominada **pop**.

```
lógico remover(P)
Entrada: pilha P
Saída: sucesso ou falha na operação

   se vazia(P) então
      retorna falso

   P.qtde ← P.qtde - 1

   retorna verdadeiro
```

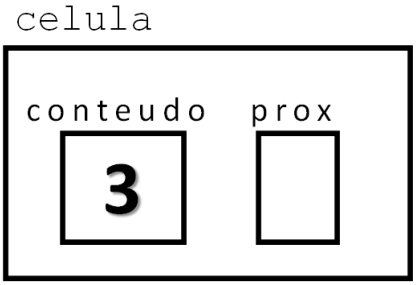
Operação de acesso

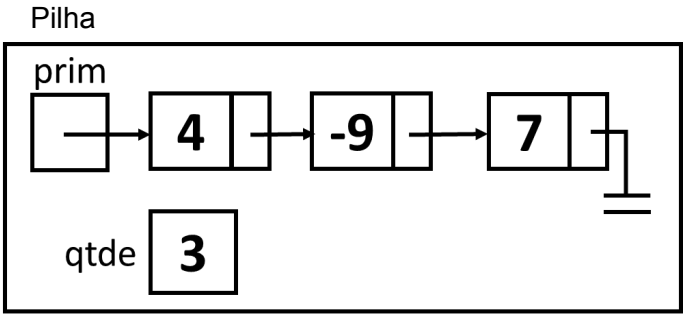
Em uma pilha **somente** temos acesso ao elemento que está no fim da lista (topo da pilha). Esta operação assume que a pilha não está vazia. O usuário dessa operação deverá testar a pilha antes de acessar. O tipo `elemento` indica um tipo genérico para informar o retorno de um elemento presente no topo da pilha. Para cada pilha, `elemento` será compatível com o tipo de dado armazenado. Esta operação somente deve ser chamada quando a pilha não estiver vazia. Comumente denominada de **top**.

```
elemento topo(P)
Entrada: pilha P
Saída: retorna o elemento que se encontra no topo da pilha

   retorna P.dados[P.qtde-1]
```

Pilhas (representação com listas encadeadas)

Representação com listas encadeadas





Observações

Assume-se que os argumentos passados para as chamadas às sub-rotinas estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.
Em uma pilha inserir e remover itens de uma mesma extremidade da lista. Desse modo, faremos inserção e remoção no início, uma vez que estas operações simplificam o acesso ao topo da pilha.

Operação auxiliar

Verificação de pilha vazia

```
lógico vazia(P)
Entrada: pilha P
Saída: verdadeiro se estiver vazia; falso caso contrário

   se P.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Na pilha usamos a **inserção no início**, logo o pseudocódigo abaixo é o mesmo desta funcionalidade. Operação comumente denominada **push**.

```
lógico inserir(P, ctd)
Entrada: pilha P, conteúdo ctd a inserir
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) então
     retorna falso

  cel.conteudo ← ctd
  cel.prox ← P.prim
  P.prim ← cel

  P.qtde ← P.qtde + 1

  retorna verdadeiro
```

Operação de remoção

Na pilha usamos a **remoção no início**. Logo, o pseudocódigo abaixo é o mesmo desta funcionalidade presente na lista encadeada. Operação comumente denominada **pop**.

```
lógico remover(P)
Entrada: pilha P
Saída: sucesso ou falha na operação

   se vazia(P) então
      retorna falso

   P.prim ← P.prim.prox
   P.qtde ← P.qtde - 1

   retorna verdadeiro
```

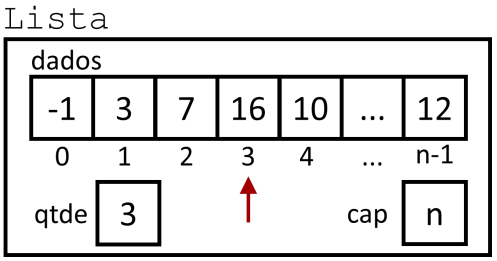
Operação de acesso

Em uma pilha **somente** temos acesso ao elemento que está no início da lista (topo da pilha). Esta operação assume que a pilha não está vazia. O usuário dessa operação deverá testar a pilha antes de acessar. O tipo `elemento` indica um tipo genérico para informar o retorno de um elemento presente no topo da pilha. Para cada pilha, `elemento` será compatível com o tipo de dado armazenado. Esta operação somente deve ser chamada quando a pilha não estiver vazia. Comumente denominada de **top**.

```
elemento topo(P)
Entrada: pilha P
Saída: retorna o elemento que se encontra no topo da pilha

   retorna P.prim.conteudo
```

Listas Lineares Ordenadas Sequenciais

Representação



Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento está na primeira posição (índice `0`) e o último no índice `qtde-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `qtde` da lista, o qual indica a próxima posição **livre**.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da lista (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.)
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos que a lista poderá conter.
- A representação acima compreende somente a parte de dados do TAD Lista, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Operações de acesso aos dados da lista

`L1.dados`: acesso ao campo/membro `dados` da lista `L1` (`dados` é um arranjo/vetor)
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L1.cap`: acesso ao campo/membro `cap` da lista `L1` (`cap` armazena o número máximo de itens que se pode armazenar na lista, ou seja, sua capacidade)
`L1.dados[i]`: acesso ao item situado no índice `i` do campo/membro `dados` da lista `L1`

Operações auxiliares

Capacidade da lista

```
inteiro capacidade(L)
Entrada: lista L
Saída: capacidade da lista

   retorna L.cap
```

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens
       presentes na lista

   retorna L.qtde
```

Verificação de lista cheia

```
lógico cheia(L)
Entrada: lista L
Saída: verdadeiro se estiver cheia; falso caso contrário

   se L.qtde = L.cap então
      retorna verdadeiro

   retorna falso
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Uma vez que os itens entrarão na lista de forma ordenada, somente uma operação de inserção será oferecida. O item será adequadamente inserido de acordo com a ordenação dos itens existentes.

```
lógico inserir(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: sucesso ou falha na operação

  se cheia(L) então
     retorna falso

  se vazia(L) então
    L.dados[0] ← ctd
  senão
    i ← L.qtde
    enquanto i > 0 e L.dados[i-1] > ctd faça
       L.dados[i] ← L.dados[i-1]
       i ← i - 1
    L.dados[i] ← ctd

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Operação de remoção

Somente uma operação de remoção está sendo apresentada nesta lista. Para inserir no início ou no fim, basta fazer k = 1 ou k = tamanho(L), respectivamente..

```
lógico remover(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação

  se vazia(L) ou invalido(k) então
     retorna falso

  para i ← k-1 até tamanho(L)-1 faça
     L.dados[i] ← L.dados[i+1]

  L.qtde ← L.qtde - 1

  retorna verdadeiro
```

Operações de busca

Busca Sequencial

```
lógico buscar(L, valor)
Entrada: lista L, valor a ser verificado na lista
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  para i ← 0 até tamanho(L)-1 faça
    se L.dados[i] = valor então
       retorna verdadeiro

  retorna falso
```

Busca Sequencial (Otimizada)

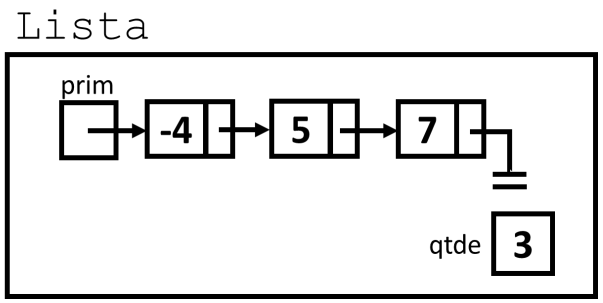
```
lógico buscar(L, valor)
Entrada: lista L, valor a ser verificado na lista
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  i ← 0
  enquanto i < tamanho(L) e valor ≤ L.dado[i] faça
     se L.dados[i] = valor então
        retorna verdadeiro
     i ← i + 1

  retorna falso
```

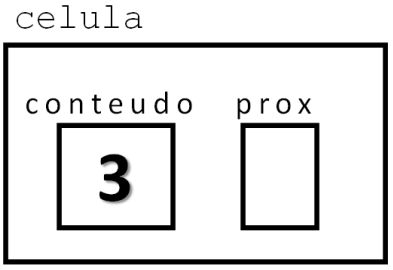
Busca Binária

```
lógico busca_binaria(L, valor)
Entrada: lista L, valor a ser verificado na lista
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  esq ← 0
  dir ← tamanho(L) - 1

  enquanto esq ≤ dir faça
    meio ← ⌊(esq + dir)/2⌋

    se L.dados[meio] = valor então
       retorna verdadeiro
    senão
       se L.dados[meio] < valor então
          dir ← meio - 1
       senão
          esq ← meio + 1

  retorna falso
```

Listas Lineares Ordenadas Encadeadas

Representação



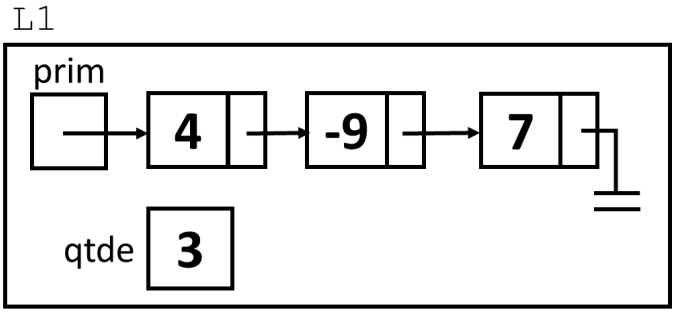


Algumas regras

- Os itens (células) da lista são acessados por meio do campo/membro `prim` da lista, sendo esse membro uma referência para o primeiro item da lista. Caso a referência seja nula, a lista estará vazia. O campo/membro `qtde` indica o número de itens presentes na lista.
- Os itens na lista não estão indexados, porém mantém uma ordem: 1ª item da lista, 2ª item da lista, etc. O acesso à primeira célula da lista é feito por meio do campo `prim`, enquanto que o acesso aos demais itens é feito utilizando as referências armazenadas no campo `prox` de cada célula.
- A representação acima compreende somente a parte de dados do TAD Lista Ordenada Encadeada, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Acesso a dados

Seja uma lista L1, conforme figura abaixo:



Operações de acesso aos dados da lista:

`L1.prim`: acesso ao campo/membro `primeiro` da lista `L1`
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L1.prim.prox`: acesso ao campo/membro `prox` da primeira célula da lista `L1`
`L1.prim.conteudo`: acesso ao campo/membro `conteudo` da primeira célula da lista `L1`

Observações

Assume-se que os argumentos passados para as chamadas às funções estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.

Operações auxiliares

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens
       presentes na lista

   retorna L.qtde
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Uma vez que os itens entrarão na lista de forma ordenada, somente uma operação de inserção será oferecida. O item será adequadamente inserido de acordo com a ordenação dos itens existentes.

```
lógico inserir(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  cel.conteudo ← ctd
  temp ← L.prim
  se vazia(L) ou temp.conteudo > ctd então
     cel.prox ← L.prim
     L.prim ← cel
  senão
     para i ← 1 .. tamanho(L) faça
        se temp.prox = NULO então
           temp.prox ← cel
           L.qtde ← L.qtde + 1
           retorna verdadeiro
        se temp.prox.conteudo > ctd então
           cel.prox ← temp.prox
           temp.prox ← cel
           L.qtde ← L.qtde + 1
           retorna verdadeiro
        temp ← temp.prox
  retorna falso
```

Operação de remoção

Somente uma operação de remoção está sendo apresentada nesta lista. Para remover no início ou no fim, basta fazer k = 1 ou k = tamanho(L), respectivamente..

```
lógico remover(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação

  se vazia(L) ou invalido(k) então
     retorna falso

  temp ← L.prim
  se L.qtde = 1 e k = 1 então
     L.prim ← NULO
  senão
     para i ← 1 até k-2 passo 1 faça
        temp ← temp.prox
     temp.prox ← temp.prox.prox

  L.qtde ← L.qtde - 1

  retorna verdadeiro
```

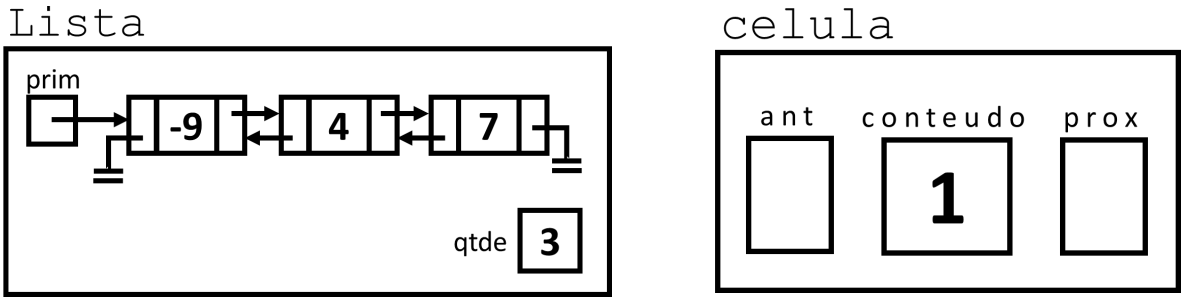
Operação de busca

Busca sequencial

```
lógico buscar(L, ctd)
Entrada: lista L, conteúdo ctd a ser buscado
Saída: sucesso ou falha na operação

  se vazia(L) ou invalido(ctd) então
     retorna falso

  temp ← L.prim
  enquanto temp ≠ NULO faça
     se temp.conteudo = ctd então
        retorna verdadeiro
     temp ← temp.prox

  retorna falso
```

Listas Lineares Duplamente Encadeadas

Representação

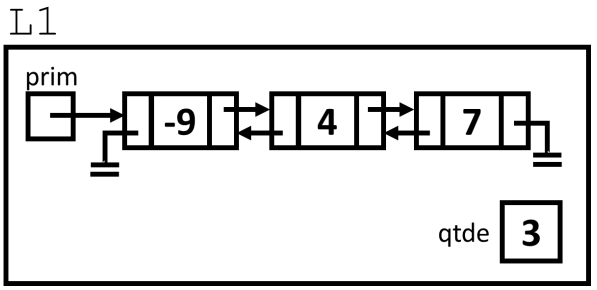


Algumas regras

- Os itens (células) da lista são acessados por meio do campo/membro `prim` da lista, sendo esse membro uma referência para o primeiro item da lista. Caso a referência seja nula, a lista estará vazia. O campo/membro `qtde` indica o número de itens presentes na lista.
- Os itens na lista não estão indexados, porém mantém uma ordem: 1ª item da lista, 2ª item da lista, etc. O acesso à primeira célula da lista é feito por meio do campo `prim`, enquanto que o acesso aos demais itens é feito utilizando as referências armazenadas nos campos `prox` ou `ant` de cada célula.
- A representação acima compreende somente a parte de dados do TAD Lista Ordenada Encadeada, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Acesso a dados

Seja uma lista L1, conforme figura abaixo:



Operações de acesso aos dados da lista:

`L1.prim`: acesso ao campo/membro `prim` da lista `L1`
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L1.prim.prox`: acesso ao campo/membro `prox` da primeira célula da lista `L1`
`temp.ant`: acesso ao campo/membro `ant` da célula referenciada pela variável `temp`
`L1.prim.conteudo`: acesso ao campo/membro `conteudo` da primeira célula da lista `L1`

Observações

Assume-se que os argumentos passados para as chamadas às funções estão adequados, de modo que somente alguns testes de validação são feitos como pré-condição das operações.

Operações auxiliares

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens
       presentes na lista

   retorna L.qtde
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se L.qtde = 0 então
      retorna verdadeiro

   retorna falso
```

Operações de inserção

Inserção no início

```
lógico inserir_inicio(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) então
     retorna falso

  cel.conteudo ← ctd
  cel.ant ← NULO
  cel.prox ← NULO

  se não vazia(L) então
     L.prim.ant ← cel

  cel.prox ← L.prim
  L.prim ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Inserção no meio

```
lógico inserir_meio(L, ctd, k)
Entrada: lista L, conteúdo ctd a ser inserido na k-ésima posição
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) ou invalido(k) então
     retorna falso

  cel.conteudo ← ctd
  cel.ant ← NULO
  cel.prox ← NULO

  se esta_vazia(L) e k = 1 então
     L.prim ← cel
     retorna verdadeiro

  temp ← L.prim
  para i ← 1 até k-2 passo 1 faça
     temp ← temp.prox

  cel.ant ← temp
  cel.prox ← temp.prox
  se temp.prox ≠ NULO então
    temp.prox.ant ← cel
  temp.prox ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Inserção no fim

```
lógico inserir_fim(L, ctd)
Entrada: lista L, conteúdo ctd a inserir
Saída: sucesso ou falha na operação
cel: célula a ser inserida

  se invalido(ctd) então
     retorna falso

  cel.conteudo ← ctd
  cel.prox ← NULO

  se vazia(L) então
     L.prim ← cel
     cel.ant ← NULO
  senão
     temp ← L.prim
     enquanto temp.prox ≠ NULO faça
        temp ← temp.prox
     cel.ant ← temp
     temp.prox ← cel

  L.qtde ← L.qtde + 1

  retorna verdadeiro
```

Operações de remoção

Remoção no início

```
lógico remover_inicio(L)
Entrada: lista L
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  se tamanho(L) então
     L.prim ← NULO
  senão
     L.prim.prox.ant ← NULO
     L.prim ← L.prim.prox

  L.qtde ← L.qtde - 1

  retorna verdadeiro
```

Remoção no meio

```
lógico remover_meio(L, k)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação

  se vazia(L) ou invalido(k) então
     retorna falso

  temp ← L.prim
  para i ← 1 até k-2 passo 1 faça
     temp ← temp.prox
  temp.prox ← temp.prox.prox
  se temp.prox ≠ NULO então
     temp.prox.ant ← temp

  L.qtde ← L.qtde - 1

  retorna verdadeiro
```

Remoção no fim

```
lógico remover_fim(L)
Entrada: lista L, k-ésima posição
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  temp ← L.prim
  se L.qtde = 1 então
     L.prim ← NULO
  senão
     enquanto temp.prox ≠ NULO faça
        temp ← temp.prox
     temp.ant.prox ← NULO

  L.qtde ← L.qtde - 1

  retorna verdadeiro
```

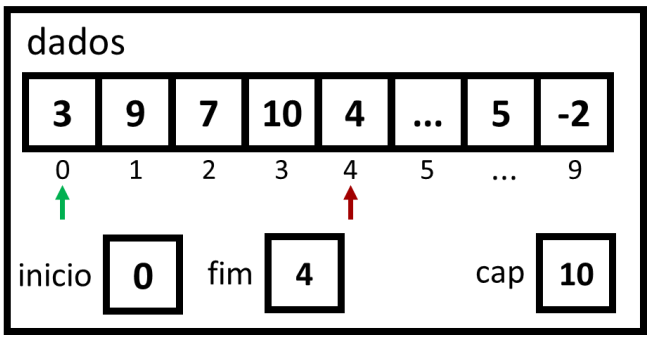
Operação de busca

Busca sequencial

```
lógico buscar(L, ctd)
Entrada: lista L, conteúdo ctd a ser buscado
Saída: sucesso ou falha na operação

  se vazia(L) ou invalido(ctd) então
     retorna falso

  temp ← L.prim
  enquanto temp ≠ NULO faça
     se temp.conteudo = ctd então
        retorna verdadeiro
     temp ← temp.prox

  retorna falso
```

Listas Circulares Sequenciais

Representação



Algumas regras

- Os elementos estão sempre contíguos na lista, sendo que o primeiro elemento está na posição de índice `inicio` e o último no índice `fim-1`.
- O marcador (seta vermelha) representa a mesma informação armazenada no campo/membro `fim` da lista, o qual indica a próxima posição **livre**.
- O marcador (seta verde) representa a mesma informação armazenada no campo/membro `inicio` da lista, o qual indica a primeira posição **ocupada**. Caso o campo `inicio` contenha o valor -1, isso indica que a lista está **vazia**.
- O membro `dados` consiste em um arranjo/vetor, sendo o espaço reservado para armazenar os dados da lista (dependendo do contexto, podem ser inteiros, reais, pessoas, etc.)
- O membro `cap` armazena a capacidade da lista, ou seja, a quantidade máxima de elementos que a lista poderá conter.
- A representação acima compreende somente a parte de dados do TAD Lista, sendo que a forma de acesso e as principais operações serão definidas em seguida.

Operações de acesso aos dados da lista

`L1.dados`: acesso ao campo/membro `dados` da lista `L1` (`dados` é um arranjo/vetor)
`L1.qtde`: acesso ao campo/membro `qtde` da lista `L1` (`qtde` armazena a quantidade de itens válidos presentes na lista)
`L1.cap`: acesso ao campo/membro `cap` da lista `L1` (`cap` armazena o número máximo de itens que se pode armazenar na lista, ou seja, sua capacidade)
`L1.dados[i]`: acesso ao item situado no índice `i` do campo/membro `dados` da lista `L1`

Operações auxiliares

Capacidade da lista

```
inteiro capacidade(L)
Entrada: lista L
Saída: capacidade da lista

   retorna L.cap
```

Tamanho da lista (quantidade de elementos válidos na lista)

```
inteiro tamanho(L)
Entrada: lista L
Saída: tamanho da lista, ou seja, a quantidade de itens
       presentes na lista

   se L.inicio = -1 então
      retorna 0

   se L.fim > L.inicio então
      retorna L.fim - L.inicio
   senão
      retorna L.fim - L.inicio + capacidade(L)
```

Verificação de lista cheia

```
lógico cheia(L)
Entrada: lista L
Saída: verdadeiro se estiver cheia; falso caso contrário

   se tamanho(L) = L.cap então
      retorna verdadeiro

   retorna falso
```

Verificação de lista vazia

```
lógico vazia(L)
Entrada: lista L
Saída: verdadeiro se estiver vazia; falso caso contrário

   se tamanho(L) = 0 então
      retorna verdadeiro

   retorna falso
```

Operação de inserção

Esta lista terá uma operação de inserção, de modo a manter os dados contíguos no vetor.

```
lógico inserir(L, ctd)
Entrada: lista L, conteúdo ctd a ser inserido
Saída: sucesso ou falha na operação

  se cheia(L) então
     retorna falso

  se vazia(L) então
    L.inicio ← 0
    L.fim ← 0
  L.dados[fim] ← ctd
  L.fim ← L.fim + 1

  se L.fim = capacidade(L) então
    L.fim ← 0

  retorna verdadeiro
```

Operação de remoção

Esta lista terá uma operação de remoção, de modo a manter os dados contíguos no vetor.

```
lógico remover(L)
Entrada: lista L
Saída: sucesso ou falha na operação

  se vazia(L) então
     retorna falso

  L.inicio ← L.inicio + 1

  se L.inicio = capacidade(L) então
     L.inicio ← 0

  se L.inicio = L.fim então
     L.inicio ← -1
     L.fim ← 0

  retorna verdadeiro
```

Algumas referências bibliográficas

GUIMARÃES, Ângelo de Moura; LAGES, Newton Alberto de Castilho. Algoritmos e estruturas de dados. Rio de Janeiro: LTC, 1994.

FARRER, et al. Algoritmos estruturados. Rio de Janeiro: LTC, 1999.

MANZANO, José Augusto N. G. Algoritmos: lógica para desenvolvimento de programação de computadores. 21 ed. São Paulo: Érica, 2007.