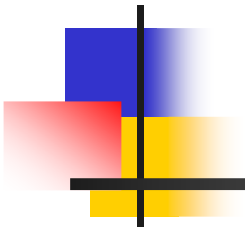
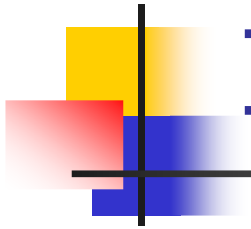


LISTAS

Profa. Michele Fúlvia Angelo



Introdução

- **Informação x Dados**

  Informação é a matéria-prima que faz com que seja necessária a existência dos computadores, pois eles são capazes de manipular e armazenar um grande volume de dados com alta performance.

  - A data 25/03: está sendo apresentado um dado;
  - Este dia é o aniversário de João: valor agregado ao dado data (informação).

- Os dados processados pelo computador são classificados de acordo com o tipo de informação que neles está contida, que basicamente são quatro tipos primitivos:
  - Inteiro;
  - Real;
  - Caracter;
  - Lógico.

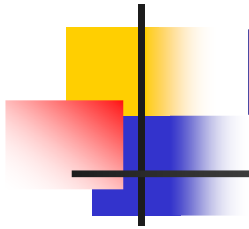
Introdução

- Os tipos primitivos não são suficiente para representar toda e qualquer informação que possa surgir.

ESTRUTURAS DE DADOS

São mecanismos de organização das informações visando um processamento eficiente das mesmas

- Vetor;
- Registros;
- **Listas**;
- Pilhas;
- Filas;
- Árvores;
- Tabelas Hash;
- Grafos;



Listas

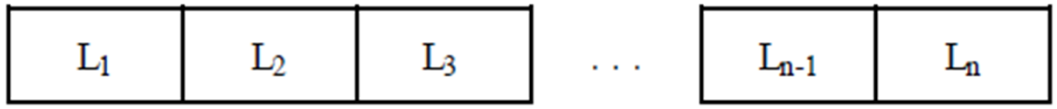
- As listas têm se mostrado um recurso bastante útil e eficiente no dia-a-dia das pessoas.
- Em computação, a lista é uma das estruturas de dados mais empregadas no desenvolvimento de programas.
- Exemplos:
  - Lista Telefônica;
  - Lista de clientes de uma agência bancária;
  - Lista de setores de disco a serem acessados por um SO;
  - Lista de pacotes a serem transmitidos em um nó de uma rede.

Listas

Definição:

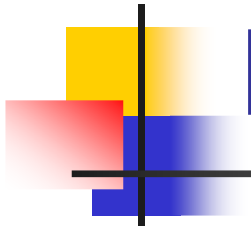
- São estruturas formadas por um conjunto de dados de forma a preservar a relação de ordem linear entre eles. Uma lista é composta por nós, os quais podem conter, cada um deles, um dado primitivo ou composto.

Representação:



Onde:

L_1: 1º elemento da lista
L_2: Sucessor de L1
L_{n-1}: Antecessor de Ln-1
L_n: Último elemento da lista.



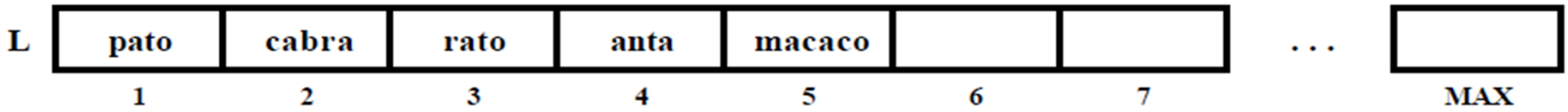
Listas

- **Operações realizadas com Listas:**
  - Criar uma lista vazia;
  - Verificar se uma lista está vazia;
  - Acessar um elemento qualquer da lista;
  - inserir um elemento numa posição específica da lista;
  - remover um elemento de uma posição específica da lista;
  - combinar duas listas em uma única;
  - particionar uma lista em duas;
  - obter cópias de uma lista;
  - determinar o total de elementos na lista;
  - ordenar os elementos da lista;
  - procurar um determinado elemento na lista;
  - apagar um determinado elemento na lista;
  - apagar uma lista;
  - outras...

Listas

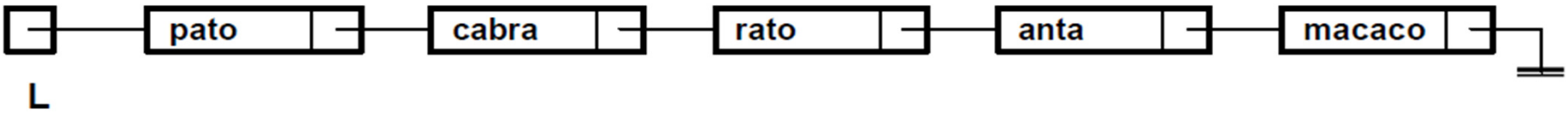
Formas de Representação:

- **Seqüencial**: Explora a seqüencialidade da memória do computador, de tal forma que os nós de uma lista são armazenados em endereços seqüenciais, ou igualmente distanciados um do outro. Pode ser representado por um vetor na memória principal.



- **Encadeada:** Esta estrutura é tida como uma seqüência de elementos encadeados por ponteiros, ou seja, cada elemento deve conter, além do dado propriamente dito, uma referência para o próximo elemento da lista.

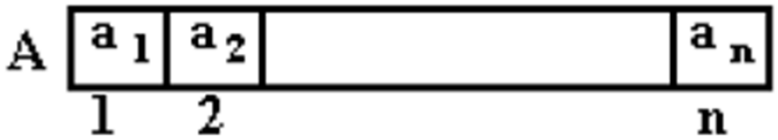
Ex: L = pato, cabra, rato, anta, macaco

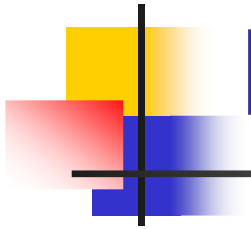


Listas Seqüenciais

Características:

- os elementos na lista estão armazenados fisicamente em posições consecutivas;
- a inserção de um elemento na posição a(i) causa o deslocamento a direita do elemento de a(i) ao último;
- a eliminação do elemento a(i) requer o deslocamento à esquerda do a(i+1) ao último;
- uma lista seqüencial ou é vazia ou pode ser escrita como (a(1), a(2), a(3), ... a(n)) onde a(i) são elementos de um mesmo conjunto A;
- a(1) é o primeiro elemento, a(i) precede a(i+1), e a(n) é o último elemento.



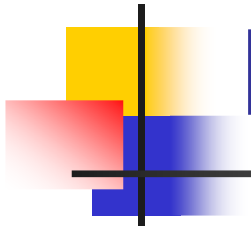


Listas Seqüenciais

As propriedades estruturadas da lista permitem responder as seguintes questões:

1. uma lista está vazia?
2. uma lista está cheia?
3. quantos elementos existem na lista?
4. qual é o elemento de uma determinada posição?
5. qual a posição de um determinado elemento?
6. inserir um elemento na lista
7. eliminar um elemento da lista

As quatro primeiras operações são feitas em tempo constante. As demais, porém, requer mais cuidados.



Listas Seqüenciais

- **Definição da ED**

```
#define MAX 10 /* tamanho máximo da lista */
typedef int telem; /* tipo base dos elementos da lista */
typedef struct
 {
    telem v[MAX]; /* vetor que contém a lista */
    int n; /* posição do último elemento da lista */
  } tlista; /* tipo lista */
```

Operações Simples utilizando Lista Seqüencial

1) Criar uma lista vazia

```
void criar (tlista *L)
{
   L->n = 0;
}
```

2) Verificar se uma lista está vazia

```
int vazia (tlista *L)
{
  return (L.n == 0);
}
```

3) Verificar se uma lista está cheia

```
int cheia (tlista *L)
{
  return (L.n == MAX);
}
```

Operações Simples utilizando Lista Seqüencial

4) Obter o tamanho de uma lista

```
int tamanho (tlista *L)
{
   return (L.n);
}
```

5) Pesquisar um dado elemento, retornando a sua posição

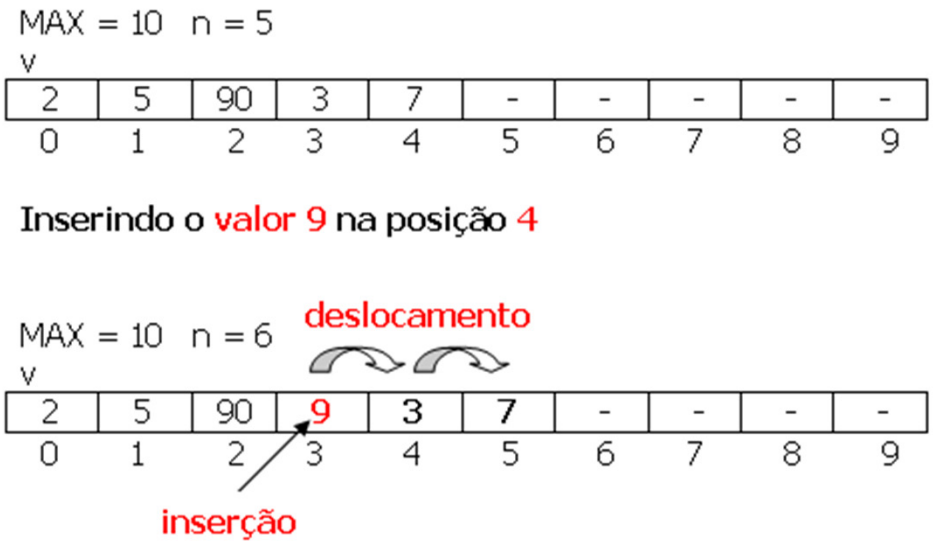
```
int posicao (tlista *L, telem dado)
{  /* Retorna a posição do elemento ou 0 caso não seja encontrado */
   int i;
   for (i=1; i<=L.n; i++)
   if (L.v[i-1] == dado)
     return (i);
   return (0);
}
```

Operações Simples utilizando Lista Seqüencial

6) Inserção de um elemento em uma determinada posição

Requer o deslocamento à direita dos elementos v(i+1)...v(n)

```
int inserir (tlista *L, int pos, telem *dado)
{
   /* Retorna 0 se a posição for inválida ou se a lista estiver cheia */
   /* Caso contrário, retorna 1 */
   int i;
   if ( (L->n == MAX)) || (pos > L->n + 1) ) return (0);
   for (i=L->n; i>=pos; i--)
     {
        L->v[i] = L->v[i-1];
      }
   L->v[pos-1] = dado;
   (L->n)++;
   return (1);
}
```

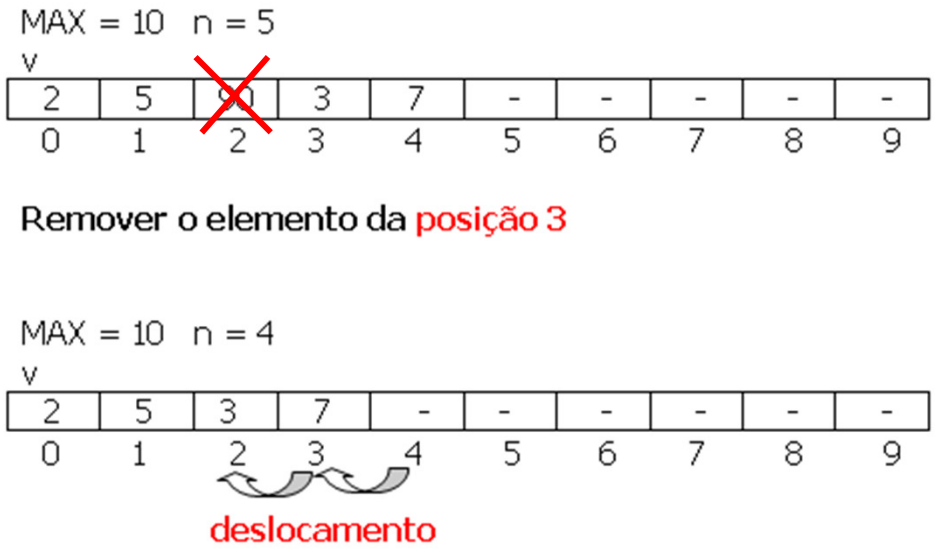


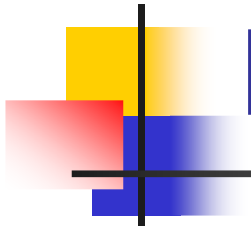
Operações Simples utilizando Lista Seqüencial

7) Remoção do elemento de uma determinada posição

Requer o deslocamento à esquerda dos elementos v(p+1)...v(n)

```
int remover (tlista *L, int pos, telem *dado)
{
     /* O parâmetro dado irá receber o elemento encontrado */
    /* Retorna 0 se a posição for inválida. Caso contrário, retorna 1 */
    int i;
    if ( (pos > L->n) || (pos <= 0) ) return (0);
   *dado = L->v[pos-1];
    for (i=pos; i<=(L->n)-1; i++)
        L->v[i-1] = L->v[i];

    (L->n)--;
    return (1);
}
```





Listas Seqüenciais

Vantagens:

- acesso direto indexado a qualquer elemento da lista;
- tempo constante para acessar o elemento *i* - dependerá somente do índice.

Desvantagens:

- movimentação quando eliminado/inserido elemento;
- tamanho máximo pré-estimado;

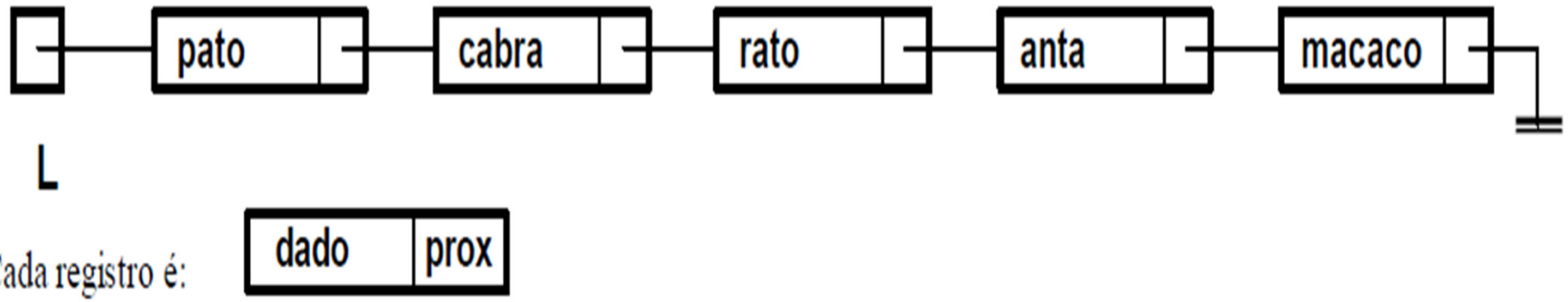
Quando usar:

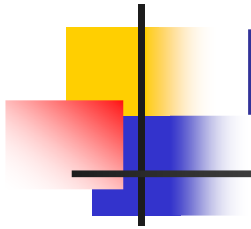
- listas pequenas;
- inserção/remoção no fim da lista;
- tamanho máximo bem definido.

Listas Encadeadas

Os elementos da lista são registros com um dos componentes destinado a guardar o endereço do registro sucessor.

Ex: L = pato, cabra, rato, anta, macaco



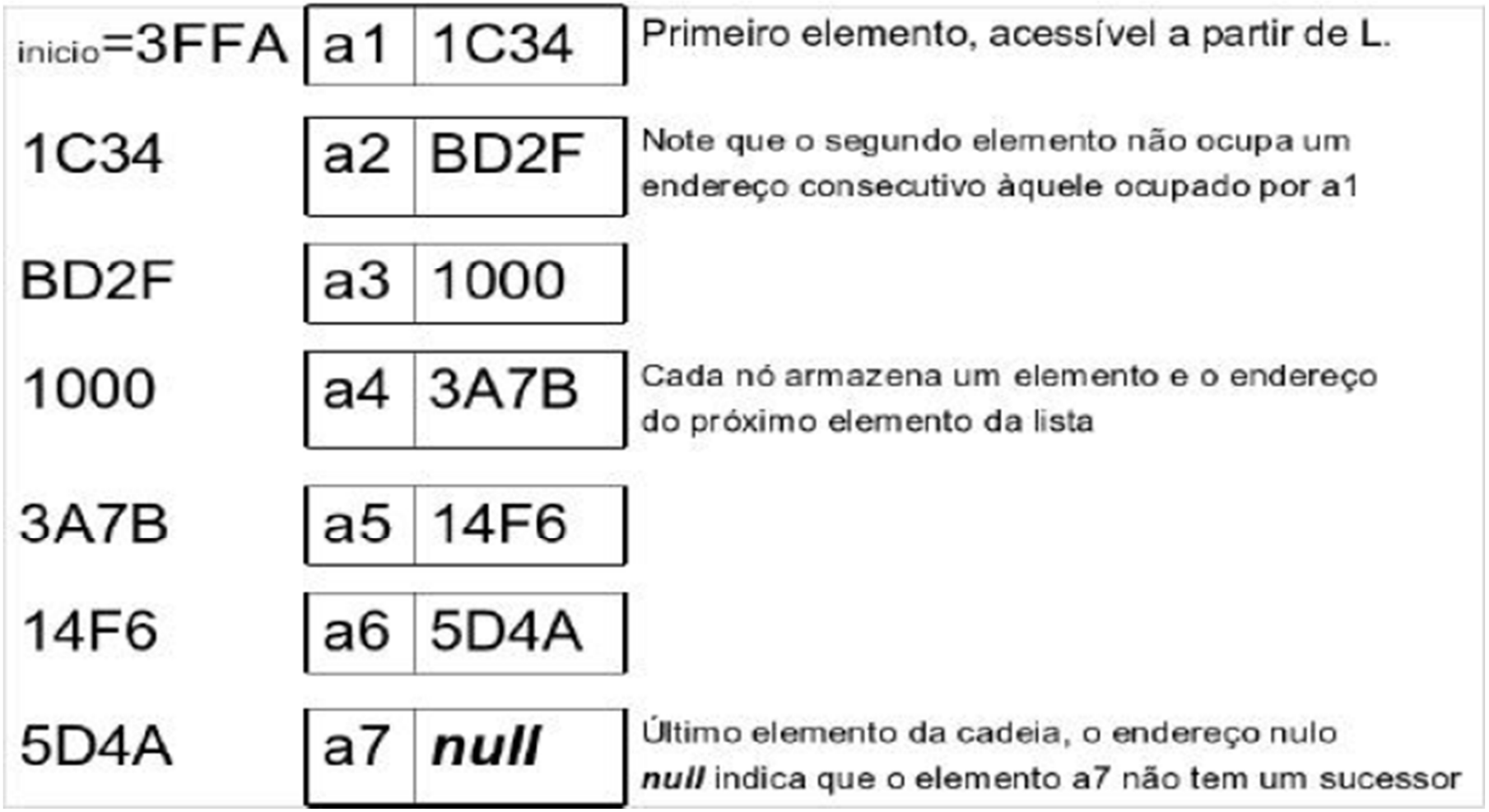


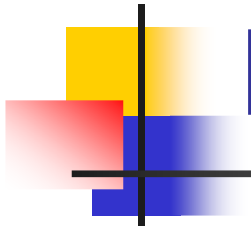
Listas Encadeadas

Características:

- Para cada novo elemento inserido na estrutura, é alocado um espaço de memória para armazená-lo.
- O espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado.
- Não se pode garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo, portanto não se tem acesso direto aos elementos da lista.
- Para que seja possível percorrer todos os elementos da lista, é necessário guardar o encadeamento dos elementos, o que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista.
- A lista é representada por um ponteiro para o primeiro elemento (ou nó)

Listas Encadeadas





Listas Encadeadas

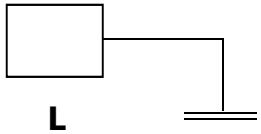
- **Definição da ED**

```
typedef int telem; /* tipo base da lista */
typedef struct no
 {
     telem dado; /* campo da informação */
     struct no* prox; /* campo do ponteiro para o próximo nó */
 } tno; /* tipo do nó */
typedef tno* tlista; /* tipo lista */
```

Operações Simples utilizando Lista Encadeada

1) Criação da lista vazia

```
void criar(tlista *L)
{
   *L = NULL;
}
```

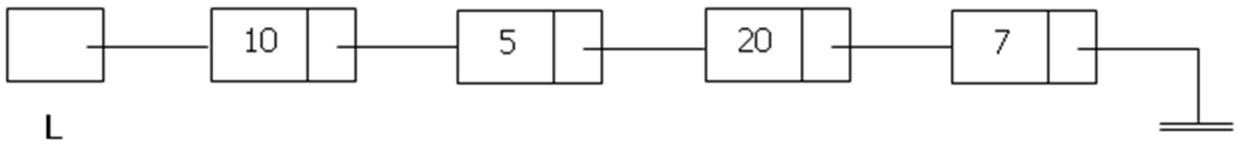


2) Verificar se a lista está vazia

```
int vazia(tlista L)
{
   return (L == NULL);
}
```

3) Obter o tamanho da lista

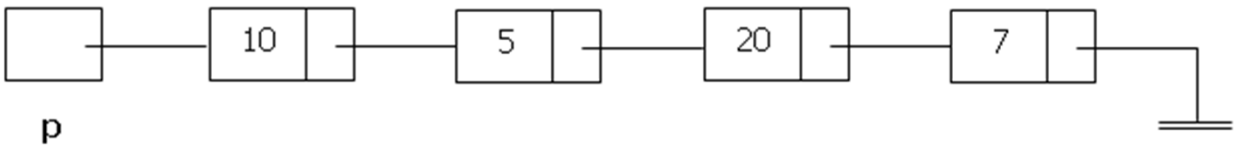
```
int tamanho(tlista L)
{  tlista p = L;
   int n = 0;
   while (p != NULL)
     {
        p = p->prox;
        n++;
     }
   return n;
}
```



Operações Simples utilizando Lista Encadeada

4) Obter o valor do elemento de uma posição dada

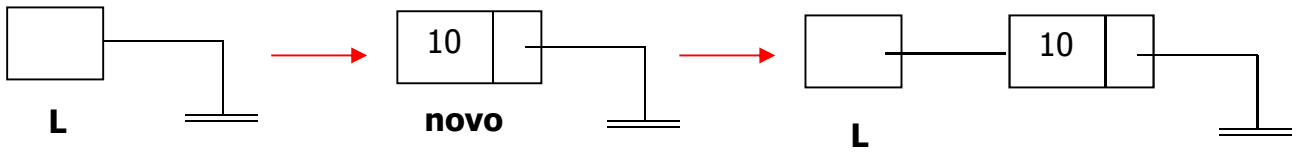
```
int verifica_elemento(tlista L, telem elem)
{
    /* Retorna 0 para lista vazia ou se o elemento não for encontrado. */
    /* Retorna 1 se o elemento for encontrado.  */
   tlista p;
   p = L;
   if (p == NULL) return 0; /* Lista vazia (underflow) */
   while ((p != NULL) && (p ->dado != elem))
    {
       p = p->prox;
     }
   if (p == NULL) return 0; /* valor não encontrado */
   return 1; /* valor encontrado */
}
```



Operações Simples utilizando Lista Encadeada

5) Inserir um elemento na lista, dada a sua posição

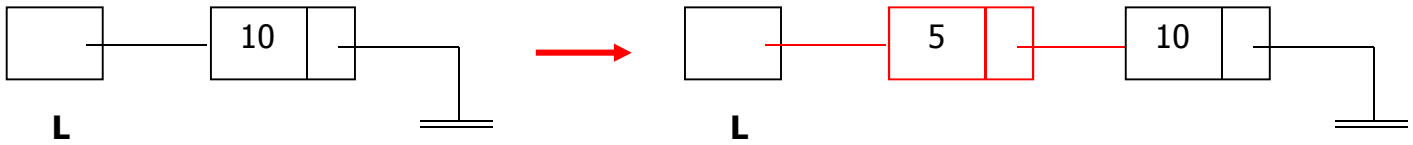
```
int inserir(tlista *L, int pos, telem valor)
{  /* Retorna 0 se a posição for inválida ou se a lista estiver cheia. Caso contrário,
      retorna 1 */

   tlista p, novo;
   int n;

   /* inserção em lista vazia */
   if (*L == NULL))
    {
        if (pos != 1) return 0; /* erro: posição inválida */
        novo = (tlista) malloc(sizeof(tno));
        if (novo == NULL) return 0; /* erro: memória insuficiente */
        novo->dado = valor;
        novo->prox = NULL;
        *L = novo;
         return 1;
    }
```



Operações Simples utilizando Lista Encadeada

5) Inserir um elemento na lista, dado a sua posição (Continuação)

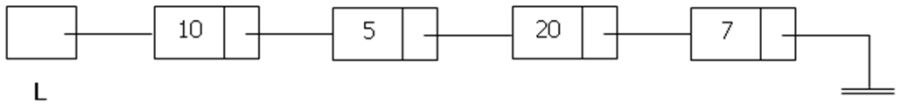
```
/* inserção na primeira posição em lista não vazia */
if (pos == 1)
  {
     novo = (tlista) malloc(sizeof(tno));
     if (novo == NULL) return 0; /* erro: memória insuficiente */
     novo->dado = valor;
     novo->prox = *L;
     *L = novo;
      return 1;
    }
```



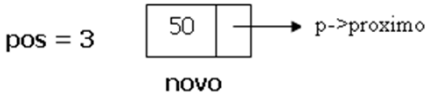
Operações Simples utilizando Lista Encadeada

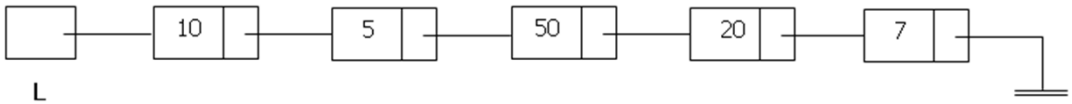
5) Inserir um elemento na lista, dado a sua posição (Continuação)

```
  /* inserção após a primeira posição em lista não vazia */
  p = *L;
  n = 1;
  while ((n <= pos-1) && (p != NULL))
   {
     p = p->prox;
     n++;
   }
   if (p == NULL) return 0; /* erro: posição inválida */
   novo = (tlista) malloc(sizeof(tno));
   if (novo == NULL) return 0; /* erro: memória insuficiente */
   novo->dado = valor;
   novo->prox = p->prox;
   p->prox = novo;
   return 1;
}
```



Inserindo o valor 50 na posição (registro) 3

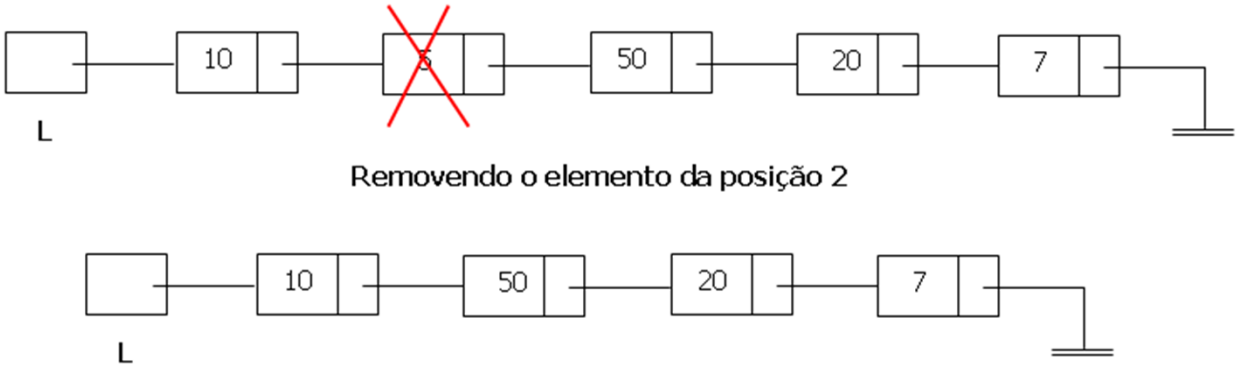


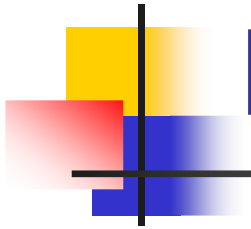


Operações Simples utilizando Lista Encadeada

6) Remover um elemento de uma determinada posição

```
int remover(tlista *L, int pos, telem *elem)
{  /* O parâmetro elem irá receber o elemento encontrado */
   /* Retorna 0 se a posição for inválida. Caso contrário, retorna 1 */
   tlista a, p;
   int n;
   if (vazia(*L)) return 0; /* erro: lista vazia */
   p = *L;
   n = 1;
   while ((n<=pos-1) && (p!=NULL))
    {
      a = p;
      p = p->prox;
      n++;
    }
   if (p == NULL) return 0; /* erro: posição inválida */
   *elem = p->dado;
   if (pos == 1)
     *L = p->prox;
   else
    a->prox = p->prox;
   free(p);
   return(1);
}
```



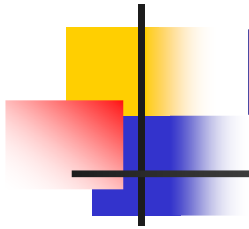


Listas Encadeadas

- **Vantagens**
  - A inserção ou remoção é feita sem que seja necessário o deslocamentos de elementos;
  - O espaço de memória a ser utilizado (para armazenar os elementos) pode ser alocado em tempo de execução;
  - O espaço alocado pode ser liberado em tempo de execução, quando não for mais necessário.

- **Desvantagens**
  - Para o acesso ao *k-ésimo* elemento da lista é necessário percorrer a lista até o elemento desejado;
  - Para verificar o tamanho da lista é necessário percorrer toda a lista;

- **Quando usar**
  - Em problemas onde não se tem uma estimativa da quantidade de elementos;
  - Problema incluir tratamento de mais de uma lista;



Referências Bibliográficas

- COLLINS, W.J. Data Structures using C And C++, 2ndedition, Prentice-Hall, 1996.
- CORMEN, T.H.; LEISERSON, C.E.; RIVEST, R.L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus.2002.
- GOODRICH, M. T.; TAMASSIA, R., Projeto de Algoritmos: Fundamentos, análise e exemplos da Internet. Bookman, Porto Alegre, 2004.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, Livros Técnicos e Científicos, 1994.
- TENEMBAUM, A.M. et all. Data Structures Using C, Prentice-Hall, 1990.
- ZIVIANI, N., Projeto de Algoritmos com Implementações em Pascal e C., Thompson, 2a. Ed, São Paulo, 2004.