Lista Dinâmica Simplesmente Encadeada

Prof^a.Dr^a.Thatyana de Faria Piola Seraphim
Prof.Dr. Enzo Seraphim

Universidade Federal de Itajubá

thatyana@unifei.edu.br
seraphim@unifei.edu.br

Listas Dinâmicas

- Uma lista é uma estrutura que armazena elementos de forma alinhada, ou seja, com elementos dispostos um após o outro.
- A lista pode ser implementada de duas formas:
  - **Estática**: todo espaço de memória a ser utilizado para armazenar os elementos, é reservado (alocado) no início da execução do programa.
  - **Dinâmica**: o espaço de memória a ser utilizado para armazenar os elementos, pode ser reservado (alocado) no decorrer da execução de um programa, quando for necessário.
- As principais características que devem ser consideradas das listas estáticas e das listas dinâmicas.

Listas Estáticas:

- Requer a especificação do tamanho máximo da lista em tempo de compilação.
- Pode disperdiçar espaço.

Listas Dinâmicas

Listas Estáticas:

- Nas operações de inserção e remoção, o tempo é proporcional ao número de elementos.
- O acesso ao elemento anterior e ao último elemento é feito em tempo constante.

Listas Dinâmicas:

- Se não é possível especificar o tamanho máximo que a lista pode atingir.
- Requer espaço para os ponteiros em cada célula.
- Nas operações de inserção e remoção, o tempo é constante.
- No acesso ao elemento anterior e ao último elemento, o tempo é proporcional ao número de elementos.

Listas Dinâmicas

As listas dinâmicas podem ser implementadas de duas formas:

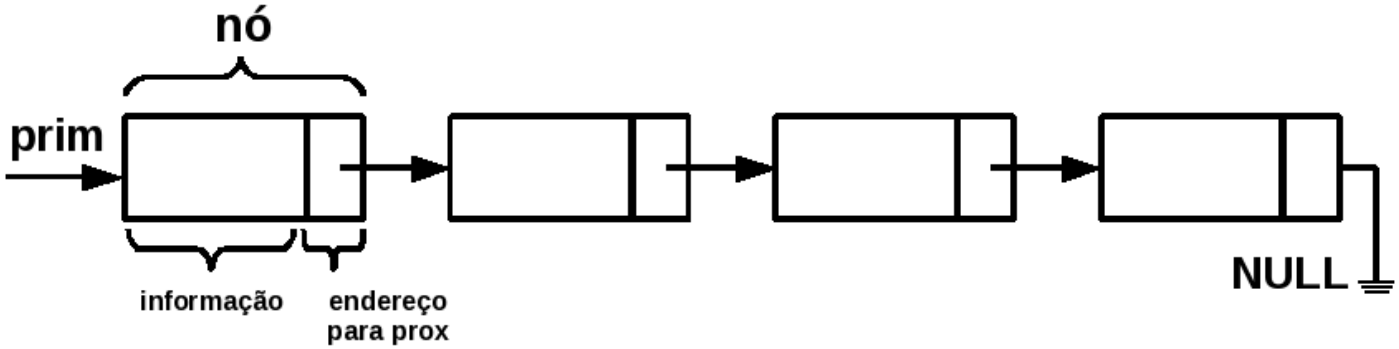
- **Lista Dinâmica Simplesmente Encadeada**: cada nó da lista possui informação do próximo nó.
- **Lista Dinâmica Duplamente Encadeada**: cada nó da lista possui informação do nó anterior e do próximo nó.

Ambas implementações usam alocação dinâmica de memória.

- Diferentemente das listas estáticas, cada nó em uma **lista dinâmica**, é alocado em qualquer região da memória (não contíguo, por isso não há acesso direto ao nó).
- A quantidade de memória utilizada para armazenar as informações é proporcional ao número de elementos.

- Na lista simplesmente encadeada, cada nó sabe apenas quem é o seu sucessor (próximo nó).
- A estrutura de um nó que forma a lista dinâmica simplesmente encadeada, deve guardar:
  - Informações que definem um elemento.

  - O endereço de memória onde está armazenado o elemento sucessor.



- Para definir um nó de uma lista simplesmente encadeada na linguagem C, são usados conceitos de *struct* e ponteiro.

Definição de um nó de valores inteiros

```
1 typedef struct no{
2   int elem;
3   struct no *prox;
4 }noDinEnc;
```

- Por convenção, quando um nó não tem próximo na lista, é usado o valor *NULL*.
- Como cada nó conhece o seu sucessor, é necessário armazenar em uma variável o endereço do primeiro nó da lista.

```
noDinEnc *prim=NULL;
```

Exercício

Declare um nó de uma lista simplesmente encadeada que armazena informações de uma agenda.

Resposta do Exercício
Definição do nó

```
1  typedef struct no{
2    char nome[40];
3    char end[50];
4    int telefone;
5    struct no *prox;
6  }noAgenda;
7
8  noAgenda *prim=NULL;
```

- Para atribuir valores em um nó de uma lista dinâmica simplesmente encadeada, primeiro o nó deve ser alocado na memória usando a função **malloc()**.

Atribuição de valores

```
noAgenda *novo = (noAgenda *)malloc(sizeof(noAgenda));
strcpy(novo->nome,"Sebastiao Jose");
strcpy(novo->end,"Rua das Flores, 35");
novo->telefone = 12345;
novo->prox = NULL;
```

Definição de um nó de valores inteiros

```
1  typedef struct no{
2    int elem;
3    struct no *prox;
4  }noDinEnc;
```

Exercício

Usando a estrutura acima, armazene na memória os 4 números {11,22,33,44} que representam uma cartela da lotofácil.

Resposta do Exercício

```
1  noDinEnc *n1=(noDincEnc *)malloc(sizeof(noDinEnc));
2  noDinEnc *n2=(noDincEnc *)malloc(sizeof(noDinEnc));
3  noDinEnc *n3=(noDincEnc *)malloc(sizeof(noDinEnc));
4  noDinEnc *n4=(noDincEnc *)malloc(sizeof(noDinEnc));
5  n1->elem=11;
6  n1->prox=n2;
7  n2->elem=22;
8  n2->prox=n3;
9  n3->elem=33;
10 n3->prox=n4;
11 n4->elem=44;
12 n4->prox=NULL;
13 //imprimindo os elementos a partir de n1
14 printf("%d", n1->elem);
15 printf("%d", n1->prox->elem);
16 printf("%d", n1->prox->prox->elem);
17 printf("%d", n1->prox->prox->prox->elem);
```

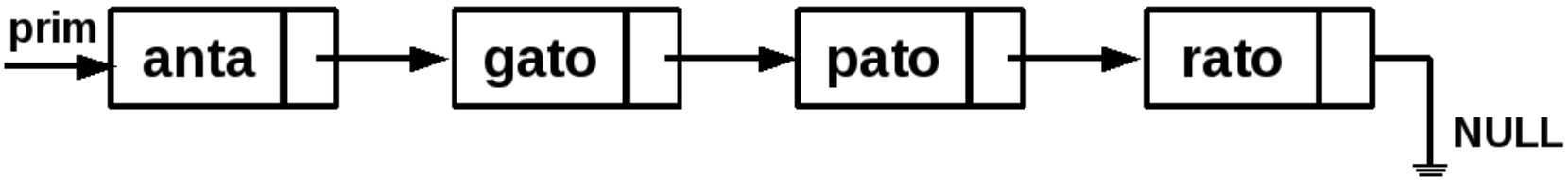
Lista dinâmica simplesmente encadeada ordenada:

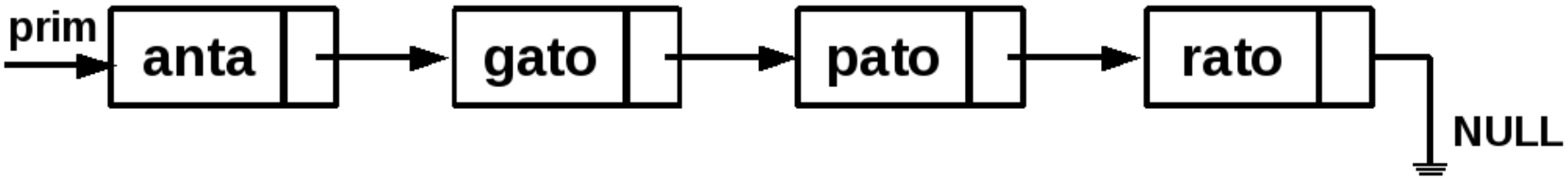
- Esta lista mantém ordenação (crescente/decrescente) entre os elementos dos nós.
- Por exemplo, uma lista ordenada de nomes de animais em ordem crescente.
- A estrutura da lista é definida abaixo:

```
typedef struct no{
  char animal[40];
  struct no *prox;
}noDinEnc;
noDinEnc *prim=NULL; //primeiro elemento da lista
```

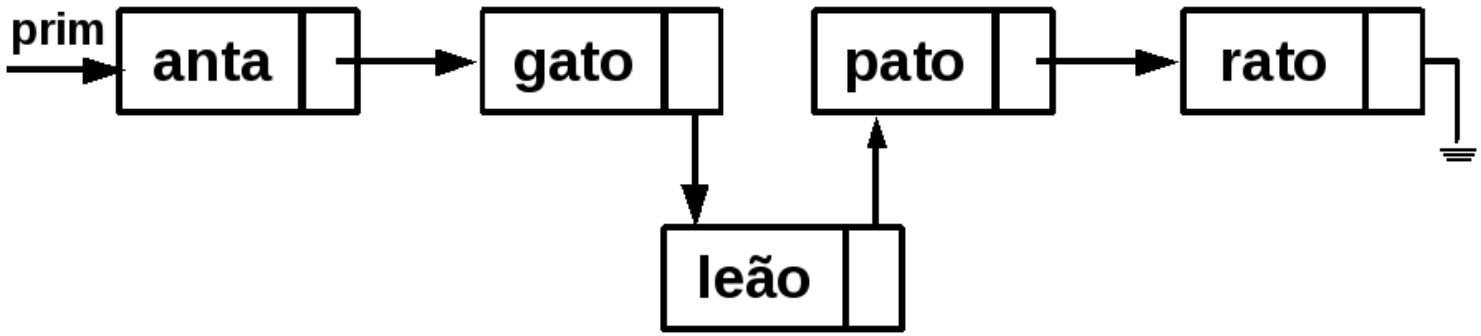
```
typedef struct no{
  char animal[40];
  struct no *prox;
}noDinEnc;
noDinEnc *prim=NULL; //primeiro elemento da lista
```

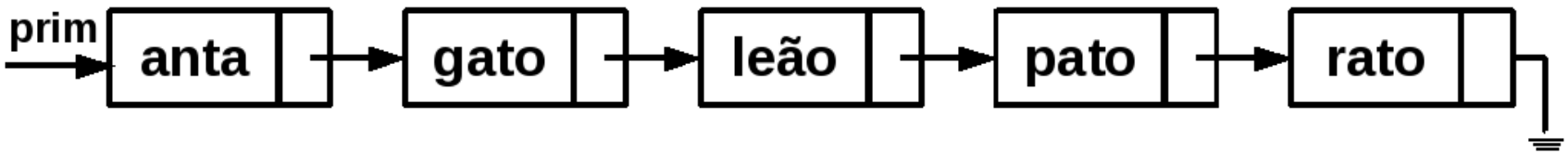
- Após a inserção dos animais pato, gato, anta, rato; a lista é representada na memória da seguinte maneira:



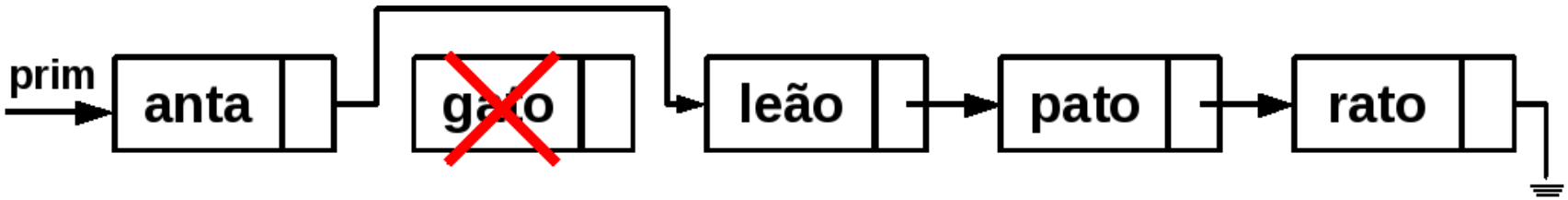


- Inserindo leão:
  - Deve-se alocar dinamicamente o espaço para o armazenamento de **leão** (**malloc(sizeof(noDinEnc))**).
  - O ponteiro **prox** de gato que antes apontava para o elemento pato, passa a apontar para o elemento leão.
  - O ponteiro **prox** de leão passa a apontar para o elemento pato.





- Removendo o nó **gato**.
- O ponteiro **prox** de anta que antes apontava para o elemento gato, passa a apontar para o elemento leão.



Características:

- Os elementos da lista no vetor não ocupam posições consecutivas.
- Para inserir um elemento na posição **(i)**:
  - Alocar espaço na memória para armazenar o novo nó.
  - Deve-se percorrer **(i-1)** nós na lista.
  - Ajustar o campo próximo do nó **(i-1)** para o novo nó inserido.
- Para remover um elemento da posição **(i)**:
  - Deve-se percorrer **(i-1)** nó na lista.
  - Ajustar o campo próximo do nó **(i-1)**.
  - Desalocar da memória o nó **(i)**.

Vantagens:

- Não requer mais a movimentação de elementos na inserção e remoção (como na lista sequencial).
- Apenas os ponteiros são alterados.
- Não é necessário reservar préviamente o tamanho máximo de elementos.

Desvantagens:

- Para acessar o elemento **(i)** é necessário percorrer $a(1), \ldots, a(i-1)$ pois, o endereço de **(i)** está disponível apenas em **(i-1)**.

As operações que podem ser realizadas dependem de cada aplicação:

- Primeiro elemento da lista.
- Último elemento da lista.
- Quantidade de elementos.
- Impressão dos elementos da lista.
- Inserção de um novo elemento.
- Remoção de um elemento.
- Pesquisa de um elemento.
- Destruir a lista.

Operações da Lista Dinâmica Encadeada

```
1  #define MAX 100 //quantidade maxima de elementos
2  typedef enum {false, true} bool;
3  typedef struct no{
4   int elem;
5   struct no *prox;
6  } noDinEnc;
7  noDinEnc *prim=NULL; //ponteiro para primeiro elemento
8  noDinEnc *ult=NULL; //ponteiro para ultimo elemento
9  int quant=0; //quantidade elementos na lista
```

Cont. Operações da Lista Dinâmica Encadeada

```
1  int primeiroListaDinEnc(); //retorna o primeiro elemento
2  int ultimoListaDinEnc(); //retorna o ultimo elemento
3  int quantListaDinEnc(); //retorna a quantidade de elementos
4  void imprimeListaDinEnc(); //impressao dos elementos
5  //insere um elemento na lista
6  bool insereListaDinEnc(int valor);
7   //remove um elemento da lista
8  bool removeListaDinEnc(int valor);
9  //retorna a posicao do elemento
10 int pesquisaListaDinEnc(int valor);
11 void destroiListaDinEnc(); //destroi a lista
```

Função main()

```
1  int main(int argc, char *argv[]){
2    int aux, i;
3    for(i=0; i<MAX; i++){
4      aux=rand() % (MAX*2);
5      if(pesquisaListaDinEnc(aux) == -1){
6        insereListaDinEnc(aux);
7      }else{
8        i--;
9      }//end else
10   }//end for
11   imprimeListaDinEnc();
```

Cont. Função main()

```
12    printf("Valor inteiro a ser procurado e removido: ");
13    scanf("%d", &aux);
14    printf("Valor %d => posicao %d\n", aux,
15            pesquisaListaDinEnc(aux));
16    removeListaDinEnc(aux);
17    imprimeListaDinEnc();
18    return 0;
19  }//end main
```

Primeiro elemento da lista

```
1  //retorna o primeiro elemento na lista ou
2  //NULL se a lista estiver vazia
3  int primeiroListaDinEnc(){
4    if(prim == NULL){
5      return 0;
6    }//end if
7    else{
8      return prim->elem;
9    }//end else
10 }//end primeiroListaDinEnc
```

Último elemento da lista

```
1  //retorna o ultimo elemento na lista ou
2  //NULL se a lista estiver vazia
3  int ultimoListaDinEnc(){
4    if(ult == NULL){
5      return 0;
6    }//end if
7    else{
8      return ult->elem;
9    }//end else
10 }//end ultimoListaDinEnc
```

Quantidade de elementos

```
1  //retorna a quantidade de elementos da lista
2  int quantListaDinEnc(){
3    return quant;
4  }//end quantListaDinEnc
```

Impressão dos elementos da lista

```
1  void imprimeListaDinEnc(){
2    int i = 0;
3    noDinEnc *atual = prim;
4    while(atual != NULL){
5      printf("[(%2d)%3d] ", i, atual->elem);
6      atual = atual->prox;
7      i++;
8      if((i%5) == 0){ //pula linha a cada 5 impressoes
9        printf("\n");
10     }//end if
11   }//end while
12   printf("\n");
13 }//end imprimeListaDinEnc
```

Inserção dos elementos

```
1  //retorna verdadeiro se inseriu o elemento na lista
2  bool insereListaDinEnc(int valor){
3    noDinEnc *ant = NULL;
4    noDinEnc *atual = prim;
5
6    //aloca o novo no na memoria
7    noDinEnc *novo=(noDinEnc *)malloc(sizeof(noDinEnc));
8
9    //encontra a posicao de insercao e
10   //quem eh seu anterior
11   while((atual!=NULL) && (atual->elem<valor)){
12     ant = atual;
13     atual = atual->prox;
14   }//end while
```

Continuação da Inserção

```
15    //insere na primeira posicao da lista
16    if(ant == NULL){
17      prim=novo;
18    }//end if
19    else{ //insere em qualquer posicao da lista
20      ant->prox = novo;
21    }//end else
22
23    //atualizando valores para o novo no
24    novo->elem = valor;
25    novo->prox = atual;
```

Continuação da Inserção

```
26    //atualizando a ultima posicao
27    if(atual == NULL){
28      ult=novo;
29    }//end if
30    quant++;
31    return true;
32  }//end insereListaDinEnc
```

Remoção dos elementos

```
1  //retorna verdadeiro se removeu o elemento na lista
2  bool removeListaDinEnc(int valor){
3    noDinEnc *ant = NULL;
4    noDinEnc *atual = prim;
5
6    //encontra a posicao de insercao
7    //e quem eh seu anterior
8    while((atual!=NULL) && (atual->elem!=valor)){
9      ant = atual;
10     atual = atual->prox;
11   }//end while
```

Continuação da Remoção

```
12      //se nao existe o elemento
13      if(atual == NULL){
14        return false;
15      }//end if
16      else{
17        //remocao na primeira posicao da lista
18        if(atual == prim){
19          prim=atual->prox;
20        }//end if
21        else{
22          //remocao em qualquer posicao da lista
23          ant->prox = atual->prox;
```

Continuação da Remoção

```
25          //remocao na ultima posicao
26          if(atual == ult){
27            ult=ant;
28          }//end if
29        }//end else
30        free(atual); //eliminando da memoria
31        quant--;
32        return true;
33      }//end else
34    }//end removeListaDinEnc
```

Pesquisa de um elemento

```
1  //retorna a posicao na lista de um elemento ou
2  //-1 quando nao encontrou
3  int pesquisaListaDinEnc(int valor){
4    int i = 0;
5    noDinEnc *atual = prim;
6    while((atual != NULL) && (atual->elem != valor)){
7      atual=atual->prox;
8      i++;
9    }//end while
10   if(atual == NULL){
11     return -1;
12   }//end if
13   else{
14     return i;
15   }//end else
16 }//end posicaoListaDinEnc
```

Destrução da lista

```
1  //destroi a lista
2  void destroiListaDinEnc(){
3    noDinEnc *atual = prim;
4    noDinEnc *apaga;
5
6    while(atual != NULL){
7      apaga = atual;
8      atual = atual->prox;
9      free(apaga);
10   }//end while
11 }//end destroiListaDinEnc
```

Problemas:

- Essa estrutura caracteriza-se por formar um encadeamento simples entre os elementos.
- Cada elemento armazena um ponteiro para o próximo elemento da lista.
- Não tem como percorrer a lista em ordem inversa de maneira eficiente.
- Dificuldade na remoção de um elemento da lista.
  - Mesmo com o ponteiro para o elemento que se deseja remover, é necessário percorrer a lista para encontrar o elemento anterior.

Solução

Lista Dinâmica Duplamente Encadeada