

# **Estruturando e Manipulando Listas em C**

WEYLER N M LOPES  
CEFET/PI

## **Objetivo**

Este trabalho tem como objetivo explicar de forma simples e gradativa a manipulação da estrutura de dados lista através da linguagem de programação C. Uma lista é uma estrutura básica muito utilizada no armazenamento de dados. Outras estruturas se baseiam em listas, como: pilhas, filas e deque. Há várias variações de listas, que abordaremos no decorrer deste trabalho. Listas duplamente encadeadas e listas circulares são algumas delas.

A nossa pretensão é que com este trabalho o aluno se habilite a utilizar a linguagem C para outras atividades na disciplina de Estrutura de Dados, familiarizando-o com conceitos como ponteiros, estruturas e alocação de memória. Dessa forma, o aluno estará apto a realizar atividades práticas, associadas às mais diversas estruturas de dados, programando nessa poderosa linguagem.

## Índice

1. Considerações Iniciais.....	01
2. Introdução .....	01
3. Lista Encadeada .....	01
3.1. Definindo um nó .....	01
3.2. Criando um nó.....	01
3.3. Definindo uma pequena lista encadeada .....	01
3.4. Definindo grandes listas encadeadas .....	01
3.5. Exercícios .....	01
4. Manipulando Listas .....	01
4.1. Contando nós .....	01
4.2. Apagando primeiro nó .....	01
4.3. Apagando qualquer nó.....	01
4.4. Acessando um determinado nó .....	01
4.5. Inserindo um novo nó .....	01
4.6. Exercícios .....	01

## 1. Considerações Iniciais

Todos os exemplos utilizados a seguir, assim como os exercícios, foram implementados e executados através do da IDE(*Integrated Development Enviroment*) LCC-32, disponível na web através do site <http://www.cs.virginia.edu/~lcc-win32/>.

Para maior simplicidade do código, não foram utilizadas bibliotecas especiais, que não estivesse em conformidade com o C ANSI.

Cada estrutura de dados, que está associada a algum exemplo ou exercício nesse trabalho, é representada através de um desenho, cujos elementos e padrões podem se vistos no apêndice 1.

Procuramos apresentar ao aluno figuras simples que o leve ao rápido entendimento do item em questão, como também uma codificação clara que permita uma rápida e fácil compreensão do código.

## 2. Introdução

A disciplina de estrutura de dados é premissa básica para o entendimento das principais formas de armazenamento, assim como, para o entendimento de vários algoritmos de acesso e manipulação de dados, muito importantes no escopo da computação atual.

Conceitos associados a listas são a base da disciplina de estrutura de dados. Entender bem aspectos estruturais e funcionais de listas é um grande indício de sucesso no entendimento de estruturas mais complexas.

Nesse contexto, a prática é fundamental. Dificilmente, algo é aprendido na computação sem uma prática efetiva. Estrutura de dados não é diferente. Para que entendamos como funciona uma lista, precisamos ver seu funcionamento em detalhes. Para conhecer sua estrutura é necessário que a montemos. É nesse sentido, de nortear o estudo a uma atividade prática, que está direcionado este trabalho.

Para que haja prática, precisamos de uma linguagem para que seja utilizada, várias foram as opções. A linguagem C foi escolhida por se adequar aos nossos principais requisitos. É uma linguagem simples, poderosa, padronizada, sintaxe fácil. Atende plenamente os nossos propósitos. Vale lembrar que o foco aqui não é a linguagem C, mas manipulação e estruturação de listas. É importante que o leitor que tenha uma boa fundamentação em C para que possa lograr o êxito que objetivamos com este trabalho.

## 3. Lista Encadeada

Uma lista encadeada é uma estrutura de dados onde cada item da lista (chamado de nó) pode possuir tipos heterogêneos de dados.

Uma lista é basicamente composta de nós e ponteiros, como descritos a seguir:

**Nó** – unidade de armazenamento de uma lista. Um nó contém dados e ponteiros;

**Ponteiro** – elemento de estruturação de uma lista. Ponteiro, também conhecido como “variável de ponteiro” é o elo de ligação entre os nós de uma lista.

Fazendo uma analogia a um trem, uma lista seria o trem. Os vagões seriam os nós. Os ponteiros seriam os elos que prendem os vagões. O termo encadeado significa ligados. Algumas referências chamam listas ligadas ao invés de listas encadeadas.

### 3.1 Definindo um nó

Existem várias formas de se definir um nó na linguagem C. Apresentaremos algumas delas.

**1. Através da definição de uma estrutura** - A estrutura define os elementos que irão compor um nó da lista. No código a seguir, está sendo definida uma estrutura que representa um nó que tem um número inteiro e um ponteiro para outro nó.

```
struct lista {  
    int num;  
    struct lista *prox;  
};
```

**2. Através da definição de um tipo de variável** - Neste caso, não é definida apenas uma estrutura, mas também um tipo de variável. O exemplo a seguir define um tipo de variável de nome `minha_lista` que representa um nó da lista.

```
typedef struct lista {  
    int num;  
    struct lista *prox;  
} minha_lista;
```

### 3.2 Criando um nó

Um nó é criado como uma variável qualquer. Assim, declaramos um nó através de uma declaração de variável. Esta declaração depende de como foi definido o nó (como mostrado no tópico anterior). Apresentaremos cada uma delas

**1. Declarando através de uma estrutura** - no exemplo a seguir um nó, de nome `no1`, está sendo declarado.

```
struct lista no1;
```

**2. Declarando através de um tipo de variável** - O exemplo a seguir tem o mesmo efeito da declaração anterior, isto é, define um nó de nome `no1`.

```
minha_lista no1;
```

Ambas declarações terão o mesmo efeito e criará a seguinte estrutura como apresentada na figura 1.

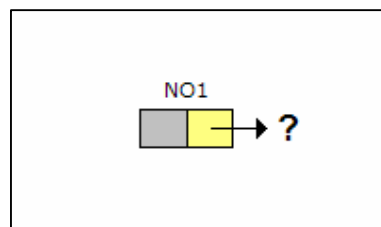


Figura 1: Nó cujo

*posição indefinida*

*ponteiro aponta pra*

A figura 1 mostra o nó criado, através de uma alocação estática. Seu ponteiro, isto é, `no1.prox`, não possui valor definido, assim, não se sabe para onde aponta.

O apêndice 1 consta de uma lista de símbolos que usaremos para representar todas as estruturas apresentadas neste trabalho.

### 3.3 Definindo uma pequena lista encadeada

Vamos agora criar uma pequena lista encadeada com apenas quatro nós.

**1. Criando os nós que irão compor a lista** – Estamos criando quatro nós: no1, no2, no3 e no4. Vale lembrar que estes nós estão sendo criados através de alocação estática.

```
minha_lista no1, no2, no3, no4;
```

**2. Criando ponteiro que apontará para o início da lista** – Este ponteiro servirá para fazer referência ao início da lista. Será usado toda vez que for necessário acessar o primeiro nó da lista.

```
minha_lista *inicio;
```

Quando estas duas instruções forem executadas no programa, as seguintes estruturas serão criadas, como mostrado na figura 2.

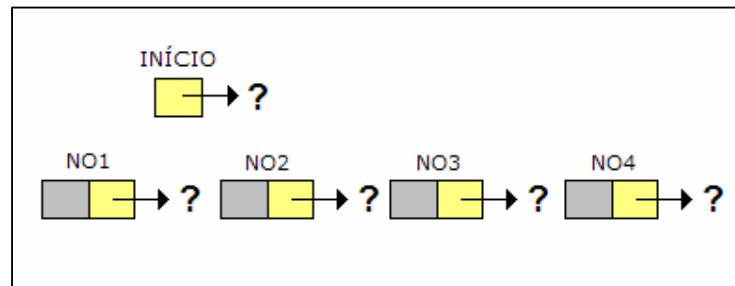


Figura 2: Nós e ponteiros não encadeados

Vale observar, na figura 2, que todos os ponteiros apontam para posições de memória indefinida. O próximo passo será ligar todas estas estruturas para que se possa criar uma lista encadeada de quatro nós.

**3. Ligando as estruturas** – Este procedimento irá ligar todos os elementos da lista, criando assim uma lista encadeada.

```
início = &no1;  
no1.prox = &no2;  
no2.prox = &no3;  
no3.prox = &no4;  
no4.prox = NULL;
```

Após a execução destas instruções, uma lista encadeada é criada, conforme figura3. Observe que agora todos os nós estão ligados e não há nenhum ponteiro apontando para posições de memória indefinidas.

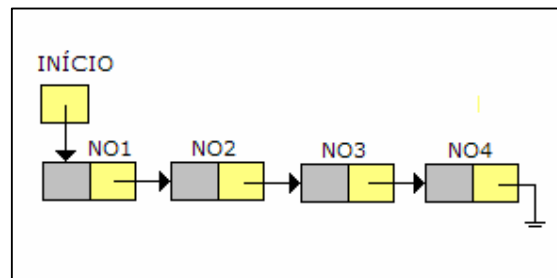


Figura 3: Lista Encadeada

### 3.4 Definindo grandes listas encadeadas

Se quisermos criar listas encadeadas grandes, isto é, com uma grande quantidade de nós, não convém que façamos uma alocação estática para cada nó. Isso tornaria o programa enorme e trabalhoso. Imagine uma lista com 20 mil nós.

Para resolver este problema, utilizaremos alocação dinâmica na criação dos nós, juntamente com estruturas de controle de repetição.

Considerando a lista de quatro nós criada anteriormente, alocaremos um nó de forma dinâmica e colocaremos este nó no início da lista.

- 1. Criar um ponteiro auxiliar** – Este ponteiro será criado para apontar para o nó que será alocado dinamicamente.

```
minha_lista *aux;  
aux = (minha_lista *)malloc(sizeof(minha_lista));
```

Após a execução destas instruções, um nó é criado, conforme figura 4.

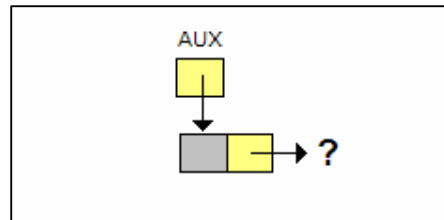


Figura 4: Nó alocado dinamicamente

## 2. Fazer o ponteiro do nó criado apontar para o início –

Dessa forma, o novo nó está sendo inserido no início da lista.

```
aux->prox = inicio;
```

Após a execução desta instrução, o nó apontará para o início da lista, conforme figura 5.

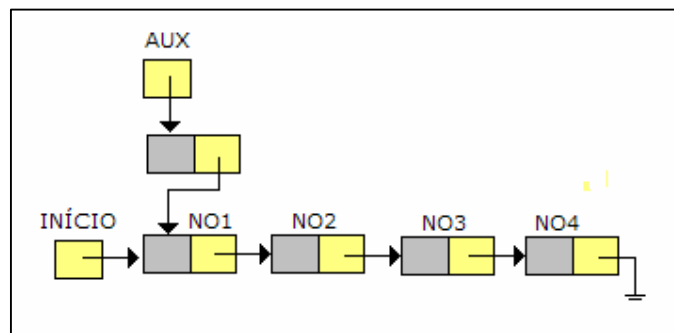


Figura 5: Um novo nó na lista - I

## 3. Fazer com que o ponteiro início aponte para o novo nó –

Início agora apontará para onde o ponteiro aux está apontando, isto é, para o novo nó.

```
inicio = aux;
```

O novo nó agora está inserido na lista, como mostrado na figura 6.

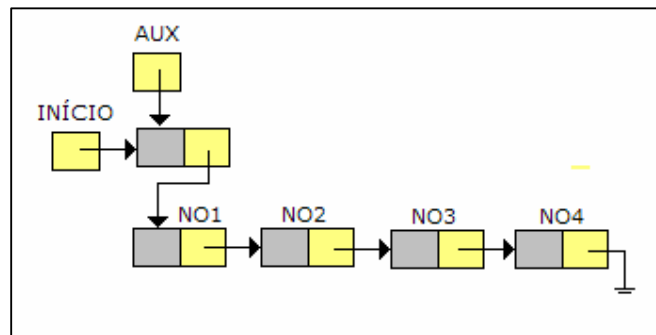


Figura 6: Um novo nó na lista - II

**4. Inserir 200 nós na lista** – Serão alocados 200 nós de forma dinâmica e inseridos à lista.

```
minha_lista *aux;  
for (int i=0; i<200; i++) {  
    aux = (minha_lista *)malloc(sizeof(minha_lista));  
    aux->prox = inicio;  
    inicio = aux;  
}
```

A figura 7 mostra a lista gerada com seus 200 nós.

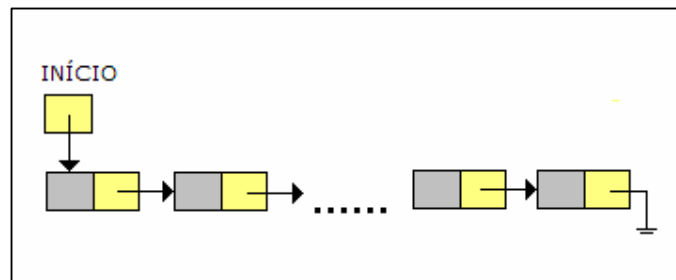


Figura 7: Lista com 200 nós

Vimos até aqui com criar listas encadeadas. Não é difícil. Esse processo consiste basicamente de criar nós (estática ou dinamicamente) e ligá-los, através de ponteiros.

Veremos a seguir a implementação de algumas funções para manipulação de listas.

### 3.5 Exercícios

- 1) O que você entende por lista encadeada no contexto das estruturas de dados?
- 2) Qual o papel das variáveis ponteiros em uma lista encadeada?
- 3) Cite vantagens de se utilizar listas ao invés de *arrays* como mecanismo de estruturação de dados.
- 4) Por que alocação dinâmica é fundamental na geração de listas encadeadas?
- 5) Qual a função que aloca dinamicamente nós de uma lista? O que esta função retorna?
- 6) Implemente uma lista de encadeada com três nós. Use alocação estática.
- 7) Implemente uma lista de encadeada com 1.000 nós. Alocação dinâmica deverá ser utilizada.
- 8) Defina um tipo de dado que represente um nó de uma lista de alunos de uma faculdade. Associado a cada aluno há um *array* de quatro ocorrências contendo os telefones do aluno.

## 4. Manipulando Listas

Este capítulo tem com objetivo apresentar como implementar as principais funções de manipulação de listas encadeadas. Funções para contar a quantidade de nós de uma lista, inserir novos nós, apagar nós, etc, serão apresentadas.

### 4.1 Contando nós

Criaremos uma função para que seja invocada toda vez que se queira contar a quantidade de nós de uma lista. A implementação de uma função para isso se deve ao fato que funções evitam a repetição de código, além de estruturar melhor o programa.

- 1. Criar o protótipo da função** – O protótipo de uma função define o nome da função e seus parâmetros de entrada e saída (valor de retorno).

```
int conta_no(minha_lista *);
```

A função de nome `conta_no` receberá como parâmetro um ponteiro para uma lista e retornará a quantidade de nós da lista.

- 2. Implementando o corpo da função** – Implementaremos o corpo da função que irá calcular o tamanho da lista.

```
int conta_no(minha_lista *p){
    int tam = 0;
    while(p){
        p = p->prox;
        tam = tam + 1;
    }
    return tam;
}
```

Essa função é simples. Ela recebe o ponteiro que aponta para o início da lista a ser contada. A estrutura de controle de repetição

“while” é executada até que o valor do ponteiro p seja NULL, isto é, zero (que representa valor lógico FALSO). A cada iteração (repetição) o ponteiro p apontará para o nó seguinte e contador tam é incrementado. A final do laço, tam terá a quantidade de nós da lista.

## 4.2 Apagando primeiro nó

Apagar o primeiro nó de uma lista é algo muito simples, não há nem a necessidade de se criar uma função para isso. Basta, em qualquer parte do programa, adicionar o seguinte código:

```
minha_lista *p = inicio;  
inicio = p->prox;  
free(p);
```

Como pode ser observado, foi criado um ponteiro p que recebe o endereço que aponta para o início da lista. O novo início da lista passa a ser o nó imediatamente posterior ao início. Finalmente, o nó o primeiro nó, que está sendo apontado por p, é apagado.

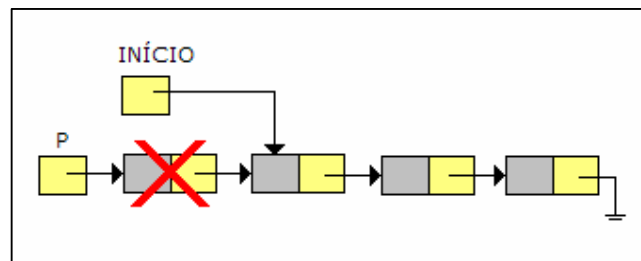


Figura 8: Lista com primeiro nó excluído

A figura 8 apresenta a lista após a exclusão de seu primeiro nó.

## 4.3 Apagando qualquer nó

Criaremos agora uma função apaga qualquer nó da lista. Para isso, essa função recebe como parâmetro o ponteiro para o início da lista e a posição do nó que será apagado.

```
minha_lista *p = inicio;  
inicio = p->prox;  
free(p);
```

Como pode ser observado, foi criado um ponteiro p que recebe o endereço que aponta para o início da lista. O novo início da lista

passa a ser o nó imediatamente posterior ao início. Finalmente, o nó o primeiro nó, que está sendo apontado por  $p$ , é apagado.

## APÊNDICE 1

### Alguns Elementos que Compõem as Estruturas



Nó cuja estrutura é composta de duas variáveis. A parte amarela representa um ponteiro.



Nó cuja estrutura é composta por três variáveis, sendo duas delas ponteiros.



Ponteiro.



Nó desalocado.



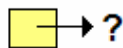
Seta que indica a posição de memória para a qual o ponteiro aponta.



Representação de um endereço indefinido.



Associação de algum ponteiro ao endereço NULL (nulo). Comumente utilizada em ponteiros de nós terminais.



Ponteiro apontando para uma posição indefinida de memória.



Ponteiro apontando para um endereço NULL (nulo).