

Flávio Augusto de Freitas
Introdução à Programação em Linguagem C/C++

<http://flavioaf.blogspot.com>

C/C++

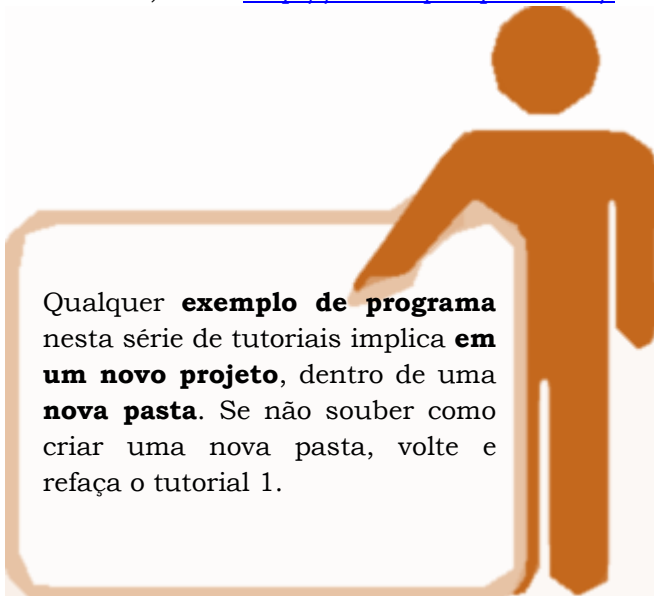
Tutorial 10 (usando Dev-C++ versão 4.9.9.2)



2011

1 INTRODUÇÃO

Esta série de tutoriais foi escrita usando o **Microsoft Windows 7 Ultimate** e o **Bloodshed Dev-C++** versão 4.9.9.2, que pode ser baixada em <http://www.bloodshed.net>. Se alguém quiser adquirir mais conhecimentos e quiser aprofundar no assunto, visite <http://www.cplusplus.com/>.



2 O QUE É UMA LISTA?


Uma lista encadeada é uma representação de uma [sequência](#) de objetos na memória do computador. Cada elemento da sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, o segundo na segunda e assim por diante.

Veja o verbete [Linked list](#) na Wikipedia.

Estrutura de uma lista encadeada

Uma **lista encadeada** (= *linked list* = lista ligada) é uma sequência de **células**; cada célula contém um objeto de algum tipo e o [endereço](#) da célula seguinte. Suporemos nesta página que os objetos armazenados nas células são do tipo **int**. A estrutura de cada célula de tal lista pode ser definida assim:

```
struct cel {  
    int conteudo;  
    struct cel *prox;  
};
```



É conveniente tratar as células como um novo [tipo-de-dados](#) e atribuir um nome a esse novo tipo:

```
typedef struct cel celula;
```

Uma célula *c* e um ponteiro *p* para uma célula podem ser declarados assim:

```
celula c;  
celula *p;
```

Se *c* é uma célula então *c.conteudo* é o conteúdo da célula e *c.prox* é o endereço da próxima célula. Se *p* é o endereço de uma célula, então [p->conteudo](#) é o conteúdo da célula e *p->prox* é o endereço da próxima célula. Se *p* é o endereço da *última* célula da lista então

p->prox vale [NULL](#).



Endereço de uma lista encadeada

O **endereço** de uma lista encadeada é o endereço de sua primeira célula. Se *p* é o endereço de uma lista, convém, às vezes, dizer simplesmente "*p* é uma lista".

Listas são animais eminentemente [recursivos](#). Para tornar isso evidente, basta fazer a seguinte observação: se *p* é uma lista então vale uma das seguintes alternativas:

- *p* == NULL ou
- *p->prox* é uma lista.

Listas com cabeça e sem cabeça

Uma lista encadeada pode ser organizada de duas maneiras diferentes, uma óbvia e outra menos óbvia.

- Lista *com* cabeça. O conteúdo da primeira célula é irrelevante: ela serve apenas para marcar o início da lista. A primeira célula é a **cabeça** (= *head cell* = *dummy cell*) da lista. A primeira célula está sempre no mesmo lugar na memória, mesmo que a lista fique vazia. Digamos que *ini* é o endereço da primeira célula. Então *ini->prox* == NULL se e somente se a lista está vazia. Para criar uma lista vazia, basta dizer

```
celula c, *ini;  
c.prox = NULL;  
ini = &c;  
OU celula *ini;  
ini = malloc (sizeof (celula));  
ini->prox = NULL;
```

- Lista *sem* cabeça. O conteúdo da primeira célula é tão relevante quanto o das demais. Nesse caso, a lista está vazia se o endereço de sua primeira célula é NULL. Para criar uma lista vazia basta fazer

```

celula *ini;
ini = NULL;

```

Suporemos no que segue que nossas listas têm cabeça. O caso de listas *sem* cabeça será tratado nos exercícios. Eu prefiro listas sem cabeça (porque são mais "puras"), mas a vida do programador fica mais fácil quando a lista tem cabeça.

Exemplos

Eis como se imprime o conteúdo de uma lista encadeada *com* cabeça:

```

// Imprime o conteúdo de uma lista encadeada
// com cabeça. O endereço da primeira célula
// é ini.

```

```

void imprima (celula *ini)
{
    celula *p;
    for (p = ini->prox; p != NULL; p = p->prox)
        printf ("%d\n", p->conteudo);
}

```

Eis a correspondente função para lista *sem* cabeça:

```

// Imprime o conteúdo de uma lista encadeada
// ini. A lista não tem cabeça.
void imprima (celula *ini)
{
    celula *p;
    for (p = ini; p != NULL; p = p->prox)
        printf ("%d\n", p->conteudo);
}

```

Busca em uma lista encadeada

Veja como é fácil verificar se um inteiro *x* pertence a uma lista encadeada, ou seja, se é igual ao conteúdo de alguma célula da lista:

```

// Esta função recebe um inteiro x e uma lista
// encadeada de inteiros. O endereço da lista é
// ini e ela tem uma celula-cabeça. A função
// devolve o endereço da celula que contém x. Se
// tal celula não existe, a função devolve NULL.

```

```

celula *busca (int x, celula *ini)
{
    celula *p;
    p = ini->prox;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}

```

Que beleza! Nada de variáveis booleanas! A função se comporta bem até mesmo quando a lista está vazia.

Eis uma versão recursiva da mesma função:

```

celula *busca2 (int x, celula *ini)
{
    if (ini->prox == NULL)
        return NULL;
    if (ini->prox->conteudo == x)
        return ini->prox;
    return busca2 (x, ini->prox);
}

```

Exercícios

1. Critique a função abaixo. Ao receber uma lista encadeada com cabeça e um inteiro *x*, ela promete devolver o endereço de uma célula com conteúdo *x*. Se tal célula não existe, promete devolver NULL.

```

celula *busca (int x, celula *ini) {
    int achou;
    celula *p;
    achou = 0;
    p = ini->prox;
    while (p != NULL && !achou) {
        if (p->conteudo == x) achou = 1;
        p = p->prox;
    }
    if (achou) return p;
    else return NULL;
}

```

2. Escreva uma versão da função busca para listas *sem* cabeça.
3. [Mínimo] Escreva uma função que encontre uma célula de conteúdo mínimo. Faça duas versões: uma iterativa e uma recursiva.
4. Escreva uma função que faça uma busca em uma lista *crescente*. Faça versões para listas com e sem cabeça. Faça versões recursiva e iterativa.
5. [Ponto médio de uma lista] Escreva uma função que receba uma lista encadeada e devolva o endereço de um nó que esteja o mais próximo possível do meio da lista. Faça isso sem contar explicitamente o número de nós da lista.
6. Verificação do tamanho. Compile e execute o seguinte programa:

```
typedef struct cel celula;
```

```
struct cel {  
    int conteudo;  
    celula *prox;  
};
```

```
int main (void) {  
    printf ("sizeof(celula) = %d\n", sizeof(celula));  
    return 0;  
}
```

Inserção em uma lista

Quero *inserir* (= *insert*) uma nova célula com conteúdo *x* entre a posição apontada por *p* e a posição seguinte [por que seguinte e não anterior?] em uma lista encadeada. É claro que isso só faz sentido se *p* é diferente de NULL.

```
// Esta função insere uma nova celula em uma  
// lista encadeada. A nova celula tem conteudo  
// x e é inserida entre a celula apontada por  
// p e a seguinte. Supõe-se que p != NULL.
```

```
void insere (int x, celula *p)  
{  
    celula *nova;  
    nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

Veja que maravilha! Não é preciso movimentar células para "criar espaço" para uma nova célula,

como fizemos para [inserir um elemento de um vetor](#). Basta mudar os valores de alguns ponteiros.

Observe também que a função se comporta corretamente mesmo quando quero inserir no fim da lista, isto é, quando *p->prox == NULL*. Se a lista tem cabeça, a função pode ser usada para inserir no início da lista: basta que *p* aponte para a célula-cabeça. Infelizmente, a função não é capaz de inserir antes da primeira célula de uma lista *sem* cabeça.

O tempo que a função consome *não depende* do ponto da lista onde quero fazer a inserção: tanto faz inserir uma nova célula na parte inicial da lista quanto na parte final. Isso é bem diferente do que ocorre com a inserção em um vetor.

Exercícios

7. Por que a seguinte versão de *insere* não funciona?

```
void insere (int x, celula *p) {  
    celula nova;  
  
    nova.conteudo = x;  
    nova.prox = p->prox;  
    p->prox = &nova;  
}
```

8. Escreva uma função que insira um novo elemento em uma lista encadeada *sem* cabeça. Será preciso tomar algumas decisões de projeto antes de começar a programar.

Remoção em uma lista

Suponha que quero [remover](#) (= *to remove* = *to delete*) uma certa célula da lista. Como posso especificar a célula em questão? A ideia mais óbvia é apontar para a célula que quero remover. Mas é fácil perceber que essa ideia não é boa. É melhor apontar para a célula *anterior* à que quero remover. Infelizmente, isso traz uma nova dificuldade: não há como pedir a remoção da *primeira* célula. Portanto, vamos nos limitar às listas com cabeça.

Vamos supor que *p* é o endereço de uma célula de uma lista com cabeça e que desejo remover a célula apontada por *p->prox*. (Note que a função de remoção não precisa saber onde a lista começa.)

```
// Esta função recebe o endereço p de uma
// célula de uma lista encadeada. A função
// remove da lista a célula p->prox. A função
// supõe que p != NULL e p->prox != NULL.
void remove (célula *p)
{
    célula *morta;
    morta = p->prox;
    p->prox = morta->prox;
    free \(morta\);
}
```

Veja que maravilha! Não é preciso copiar informações de um lugar para outro, como fizemos para [remover um elemento de um vetor](#): basta mudar o valor de um ponteiro. A função consome sempre o mesmo tempo, quer a célula a ser removida esteja perto do início da lista, quer esteja perto do fim.

Exercícios

9. Critique a seguinte versão da função remove:

```
void remove (célula *p, célula *ini) {
    célula *morta;
    morta = p->prox;
    if (morta->prox == NULL) p->prox = NULL;
    else p->prox = morta->prox;
    free \(morta\);
}
```

10. Invente um jeito de remover uma célula de uma lista encadeada *sem* cabeça. (Será preciso tomar algumas decisões de projeto antes de começar a programar.)

Mais exercícios

11. Escreva uma função que copie um vetor para uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.
12. Escreva uma função que copie uma lista encadeada para um vetor. Faça duas versões: uma iterativa e uma recursiva.
13. Escreva uma função que faça uma *cópia* de uma lista dada.
14. Escreva uma função que *concatena* duas listas encadeadas (isto é, "amarra" a segunda no fim da primeira).
15. Escreva uma função que *conta* o número de células de uma lista encadeada.

16. Escreva uma função que remove a k-ésima célula de uma lista encadeada sem cabeça. Escreva uma função que insere na lista uma nova célula com conteúdo x entre a k-ésima e a k + 1-ésima células.

17. Escreva uma função que verifica se duas listas dadas são *iguais*, ou melhor, se têm o mesmo conteúdo. Faça duas versões: uma iterativa e uma recursiva.

18. Escreva uma função que *desaloca* (função free) todos os nós de uma lista encadeada. Estamos supondo, é claro, que cada nó da lista foi originalmente alocado por malloc.

19. Escreva uma função que *inverte* a ordem das células de uma lista encadeada (a primeira passa a ser a última, a segunda passa a ser a penúltima etc.). Faça isso sem usar espaço auxiliar; apenas altere os ponteiros. Dê duas soluções: uma iterativa e uma recursiva.

20. **Projeto de Programação.** Digamos que um *texto* é um vetor de caracteres contendo apenas letras, espaços e sinais de pontuação. Digamos que uma *palavra* é um segmento maximal de texto que consiste apenas de letras. Escreva uma função que recebe um texto e imprime uma relação de todas as palavras que ocorrem no texto juntamente com o número de ocorrências de cada palavra.

Outros tipos de listas

A partir de agora, tudo é festa: você pode inventar uma grande variedade de tipos de listas encadeadas. Por exemplo, você pode fazer uma lista encadeada **circular**: a última célula aponta para a primeira. A lista pode ou não ter uma célula-cabeça (você decide). Para especificar uma lista circular, basta fornecer um endereço (por exemplo, o endereço da última célula).

Outro tipo útil é a lista **duplamente encadeada**: cada célula contém o endereço da célula anterior e o da célula seguinte. A lista pode ou não ter uma célula-cabeça (você decide). A lista pode até ter uma célula-rabo se você achar isso útil!

Pense nas seguintes questões, apropriadas para qualquer tipo de lista encadeada. Em que condições a lista está vazia? Como remover a célula apontada por p? Idem para a célula seguinte à apontada por p? Idem para a célula anterior à apontada por p? Como inserir uma

nova célula entre o elemento apontado por p e o seu antecessor? Idem entre p e seu sucessor?

Exercícios

21. Descreva, em linguagem C, a estrutura de uma das células de uma lista duplamente encadeada.
22. Escreva uma função que remove de uma lista duplamente encadeada a célula apontada por p. (Que dados sua função recebe? Que coisa devolve?)
23. Escreva uma função que insira em uma lista duplamente encadeada, logo após a célula apontada por p, uma nova célula com conteúdo y. (Que dados sua função recebe? Que coisa devolve?)
24. **Problema de Josephus.** Imagine que temos n pessoas dispostas em círculo. Suponha que as pessoas estão numeradas 1 a n no sentido horário. Começando com a pessoa de número 1, percorra o círculo no sentido horário e elimine cada m-ésima pessoa enquanto o círculo tiver duas ou mais pessoas. Qual o número do sobrevivente?

Busca-e-remoção

Suponha que ini é o endereço de uma lista encadeada *com* cabeça. Nosso problema: Dado um inteiro y, remover da lista a primeira célula que contém y (se tal célula não existe, não é preciso fazer nada).

```
// Esta função recebe uma lista encadeada ini,  
// com cabeça, e remove da lista a primeira  
// célula que contiver y, se tal célula existir.
```

```
void buscaEremove (int y, celula *ini)  
{  
    celula *p, *q;  
    p = ini;  
    q = ini->prox;  
    while (q != NULL && q->conteudo != y) {  
        p = q;  
        q = q->prox;  
    }  
    if (q != NULL) {  
        p->prox = q->prox;  
        free (q);  
    }  
}
```

Invariante: no início de cada iteração (imediatamente antes da comparação de q com NULL), temos

q == p->prox ,

ou seja, q está sempre um passo à frente de p.

Exercícios

25. Escreva uma função busca-e-remove para listas encadeadas *sem* cabeça (só pra ver que dor de cabeça isso dá).

Busca-e-inserção

Mais uma vez, suponha que tenho uma lista encadeada ini, com cabeça. (É óbvio que ini é diferente de NULL.) Nosso problema: Inserir na lista uma nova célula com conteúdo x imediatamente *antes* da primeira célula que tiver conteúdo y; se tal célula não existe, inserir x no *fim* da lista.

```
// Esta função recebe uma lista encadeada ini,  
// com cabeça, e insere na lista uma nova célula  
// imediatamente antes da primeira que  
// contiver y.  
// Se nenhuma célula contém y, insere a nova  
// célula no fim da lista. O conteúdo da nova  
// célula é x.
```

```
void buscaEinsere (int x, int y, celula *ini)  
{  
    celula *p, *q, *nova;  
    nova = mallocX (sizeof (celula));  
    nova->conteudo = x;  
    p = ini;  
    q = ini->prox;  
    while (q != NULL && q->conteudo != y) {  
        p = q;  
        q = q->prox;  
    }  
    nova->prox = q;  
    p->prox = nova;  
}
```

Exercícios

26. Escreva uma função busca-e-insere para listas encadeadas *sem* cabeça (só pra ver que dor de cabeça isso dá).
27. Escreva uma função para remover de uma lista encadeada todos os elementos que contêm y.

3 PROGRAMA-EXEMPLO

```
#include <cstdlib>
#include <iostream>
#include <string.h>
#include <windows.h>

using namespace std;

#define BUFFER 64

/* Estrutura da lista declarada para armazenar
nossos dados. */
typedef struct lista {
    char *nome;
    int idade;
    struct lista *proximo;
} Dados;

/* Prototipo das funcoes de manuseio dos dados.
*/
Dados *inicia_dados(char *nome, int idade);
Dados *insere_dados(Dados *dados, char *nome,
int idade);
void exibe_dados(Dados *dados);
void busca_dados(Dados *dados, char *chave);
Dados *deleta_dados(Dados *dados);
int checa_vazio(Dados *dados);

/* Prototipo das funcoes do menu.*/
void insere(void);
void exibe(void);
void busca(void);
void deleta(void);

/* Inicializa a estrutura de dados principal. */
Dados *principal = NULL;

/* Cria a nova lista apontando o proximo no para
NULL. */
Dados *inicia_dados(char *nome, int idade) {
    Dados *novo;

    novo = (Dados *)malloc(sizeof(Dados));
    novo->nome = (char *)malloc(strlen(nome)+1);
    strncpy(novo->nome, nome, strlen(nome)+1);
    novo->idade = idade;
    novo->proximo = NULL;

    return novo;
}
```

```
/* Como a lista nao esta mais vazia, apontamos o
proximo no para lista anterior. */
Dados *insere_dados(Dados *dados, char *nome,
int idade) {
    Dados *novo;

    novo = (Dados *) malloc(sizeof(Dados));
    novo->nome = (char *) malloc(strlen(nome)+1);
    strncpy(novo->nome, nome, strlen(nome)+1);
    novo->idade = idade;
    novo->proximo = dados;

    return novo;
}

/* Percorre todos os campos da lista e imprime
ate o ponteiro proximo chegar em NULL. */
void exibe_dados(Dados *dados) {
    printf("Cadastro:\n\n");
    printf("-----\n");
    for (; dados != NULL; dados = dados->proximo) {
        printf("Nome: %s\n", dados->nome);
        printf("Idade: %d\n", dados->idade);
        printf("-----\n");
    }

    getchar();
}

/* Percorre cada ponta comparando o nome com
a chave. */
void busca_dados(Dados *dados, char *chave) {
    int achou = 0;

    printf("Cadastro:\n\n");
    for (; dados != NULL; dados = dados->proximo) {
        if (strcmp(chave, dados->nome) == 0) {
            printf("-----\n");
            printf("Nome: %s\n", dados->nome);
            printf("Idade: %d\n", dados->idade);
            printf("-----\n");
            achou++;
        }
    }

    if (achou == 0)
        printf("Nenhum resultado encontrado.\n");
    else
        printf("Encontrados %d registros.\n", achou);
    Sleep(1);
}
```



```

/* Deleta o ultimo registro inserido. */
Dados *deleta_dados(Dados *dados) {
    Dados *novo;

    novo = dados->proximo;
    free(dados->nome);
    free(dados);

    printf("O ultimo registro inserido foi deletado
com sucesso.\n");
    Sleep(1);

    return novo;
}

/* Apenas checa se a lista e NULL ou nao. */
int checa_vazio(Dados *dados) {
    if (dados == NULL) {
        printf("Lista vazia!\n");
        Sleep(1);
        return 1;
    } else
        return 0;
}

/* Obtem os dados necessarios para chamar as
funcoes de manuseio de dados. */
void insere(void) {
    char *nome;
    int idade;

    nome = (char *)malloc(BUFFER);
    printf("\n\nDigite o Nome: \n----> ");
    scanf("%s", nome);
    printf("\n");

    printf("Digite a Idade: \n----> ");
    scanf("%d", &idade);
    printf("\n");

    if (principal == NULL)
        principal = inicia_dados(nome, idade);
    else
        principal = insere_dados(principal, nome,
idade);
}

void exibe(void) {
    if (!checa_vazio(principal))
        exibe_dados(principal);
}

```

```

void busca(void) {
    char *chave;

    if (!checa_vazio(principal)) {
        chave = (char *)malloc(BUFFER);
        printf("Digite o nome para buscar: \n--> ");
        scanf("%s", chave);
        busca_dados(principal, chave);
    }
}

void deleta(void) {
    if (!checa_vazio(principal))
        principal = deleta_dados(principal);
}

int main(void) {
    char escolha;

    do {
        system("CLS");
        printf("\n\t\tCadastro de Pessoas\n\n");
        printf("Escolha uma opcao: \n");
        printf("1 - Insere Dados\n");
        printf("2 - Exibe Dados\n");
        printf("3 - Busca Dados\n");
        printf("4 - Deleta Dados\n");
        printf("5 - Sair\n\n");

        scanf("%c", &escolha);
        switch(escolha) {
            case '1':
                insere(); break;
            case '2':
                exibe(); break;
            case '3':
                busca(); break;
            case '4':
                deleta(); break;
            case '5':
                exit(0); break;
            default:
                printf("Digite uma opcao valida!\n");
                Sleep(1);
                break;
        }

        getchar();
    } while (escolha > 0); /* Loop Principal. */

    return 0;
}

```


4 EXERCÍCIOS PROPOSTOS

- a) Implemente uma lista que armazene dados de pessoas, como nome, endereço, data de nascimento, telefone etc., ou seja, uma agenda de contatos.
- b) Implemente uma lista que armazene dados de localização geográfica de pontos turísticos, como nome do local, latitude, norte ou sul, longitude, leste ou oeste.

5 TERMINAMOS

Terminamos por aqui. O que está esperando, saia do Dev-C++ e corra para pegar o próximo tutorial em <http://flavioaf.blogspot.com>. Lá você encontra também a opção seguir. Seguindo o blog você se mantém sempre atualizado de qualquer lançamento novo.