

Unidade 1

Seção 1

Acesse este conteúdo
pelo smartphone



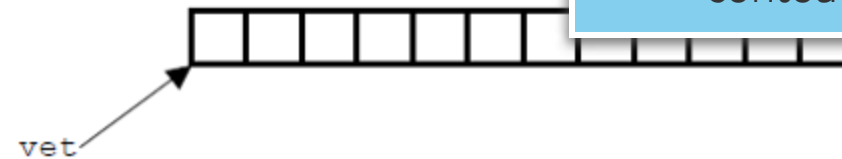
O que é isso?
Clique no código e saiba
mais.

Algoritmos e Estrutura de Dados

Webaula 1

Listas ligadas

Segundo Celes (2004), a utilização do vetor para representação de um conjunto de dados é a forma primitiva de representação existente, em que podemos definir um tamanho máximo de elementos a ser utilizado nele. O uso do vetor, ao ser de memória contíguo para armazenar seus elementos e assim torná-los acessíveis a partir do primeiro e por meio de um ponteiro.



Fonte: Celes, W., Cerqueira, R., Rangel, J. L. (2004, p. 134).

Ponteiro

Um ponteiro é um tipo de variável que armazena um endereço de memória e não o conteúdo da posição de memória.

Listas Ligadas

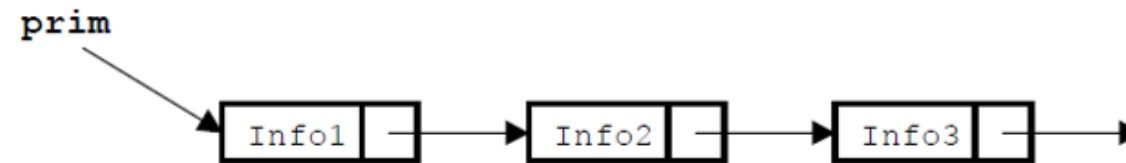
Uma Lista Ligada, também conhecida como Lista Encadeada, é um conjunto de dados dispostos por uma sequência de nós, cuja relação de sucessão desses elementos é determinada por um ponteiro que indica a posição do próximo elemento, podendo estar ordenado ou não (Silva, 2007).

Uma lista é uma coleção

$L:[a_1, a_2, \dots, a_n]$

em que $n > 0$ e sua propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente.

O nó de lista é composto por duas informações, uma informação armazenada e um ponteiro que indica o próximo elemento da lista.



Fonte: Celes, W., Cerqueira, R., Rangel, J. L. (2004, p. 135).

Diferente dos vetores, em que o armazenamento é realizado de forma contígua, a Lista Ligada estabelece a sequência de forma lógica.

Na lista encadeada é definido um ponto inicial ou um ponteiro para o começo da lista, e a partir daí pode se inserir elementos, remover ou realizar buscas na lista.

Quando uma lista está sem nós, é então definida como lista vazia ou lista nula, assim, o valor do ponteiro para o próximo nó da lista é considerado como ponteiro nulo.

Procedimentos básicos de manipulação de listas:

Criação ou definição da estrutura de uma lista

Inicialização da lista

Inserção com base em um endereço como referência

Alocação de um endereço de nó para inserção na lista

Remoção do nó com base em um endereço como referência

Deslocamento do nó removido da lista

Criação ou definição da estrutura da lista

Explore a galeria e conheça como criar uma estrutura de lista.

Ao criar uma estrutura de uma lista, definimos também o tipo de dados que será utilizado em sua implementação.

Os elementos de informação de uma lista podem ser do tipo int, char e/ou float.

```
struct lista {  
    int info;  
    struct lista* prox;  
};  
typedef struct lista Lista;
```

Fonte: elaborada pelo autor (2017).



Inicialização da lista

Precisamos inicializar a lista para a utilizarmos após sua criação.

Para isso, uma das formas possíveis é criar uma função em que inicializamos a lista como nula.

```
/* Função para inicialização: retorna uma  
lista vazia */  
Lista* inicializa (void)  
{  
    return NULL;  
}
```

Ponteiros – elementos de ligação

A utilização de ponteiros é indicada nos casos em que é preciso conhecer o endereço onde está armazenada a variável.

`int *ptr; /* sendo um ponteiro do tipo inteiro*/`

`float *ptr; /* sendo um ponteiro do tipo ponto flutuante*/`

`char *ptr; /* sendo um ponteiro do tipo caracteres*/`

```
/* Exemplo de uma estrutura de lista
declarada para armazenar dados de uma
agenda. */
```

```
typedef struct lista {
    char *nome; /*Declaração de um
ponteiro do tipo char
    int telefone;
    struct lista *proximo;
} Dados;
```

Para sabermos o endereço da memória reservada para a variável, utiliza-se o operador & junto do nome de uma variável.

```
int x = 10; /*variável  
int *p;    /*ponteiro  
p = &x;    /*ponteiro p aponta para o endereço da variável x
```

Em listas, além do uso de ponteiros, utilizamos também as alocações dinâmicas de memória, que designam porções de memórias para que as listas as utilizem.

Para isso utilizamos a função `malloc()`, que é a responsável pela reserva de espaços na memória principal. Sua finalidade é alocar uma faixa de bytes consecutivos na memória do computador e retornar seu endereço ao sistema.

Exemplo de utilização da função `malloc()` e de ponteiro:

```
char *pnt;  
pnt = malloc (2); /* Aloca 2 bytes na  
memória */  
scanf ("%c", pnt);  
O endereço retornado é o da posição  
inicial onde se localiza os bytes alocados.
```

Em uma lista, precisamos alocar o tipo de dado no qual foram declarados os elementos dela, e por esse tipo de dados ocupar vários bytes na memória, precisaremos utilizar a função `sizeof`, que permite nos informar quantos bytes o tipo de elemento criado terá.

Exemplo de programa em C com `malloc()` e `sizeof`

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    int *p;
    p=(int *) malloc(sizeof(int));

    if (!p) {
        printf("Erro de memoria insuficiente");
    }else{
        printf("Memoria alocada com sucesso");
    }

    return 0;
}
```

Fonte: elaborado pelo autor (2017)

Podemos classificar uma lista por meio de um cabeçalho, cujo primeiro elemento permanece sempre como ponto inicial na memória, independentemente da lista ter valores inseridos ou não. Assim, start é o endereço inicial da nossa lista e para determinar que ela esteja vazia, insere-se a informação NULL no próximo elemento.

```
celula *start;  
start = malloc (sizeof (celula));  
start -> prox = NULL;
```

Outra classificação possível é a lista sem cabeçalho, cujo conteúdo do primeiro elemento tem a mesma importância que os demais. Assim, a lista é considerada vazia se o primeiro elemento for NULL. A criação desse tipo de lista vazia pode ser definida por start = NULL.

Exemplo de um trecho de uma função de tipo inteiro para inserir na lista pacientes com prioridades.

```
int insere_com_prioridade(Fila *F ,  
    Paciente *P){  
    Lista *ptnodo, *aux , *ant;  
    ptnodo =  
    (Lista*)malloc(sizeof(Lista));
```

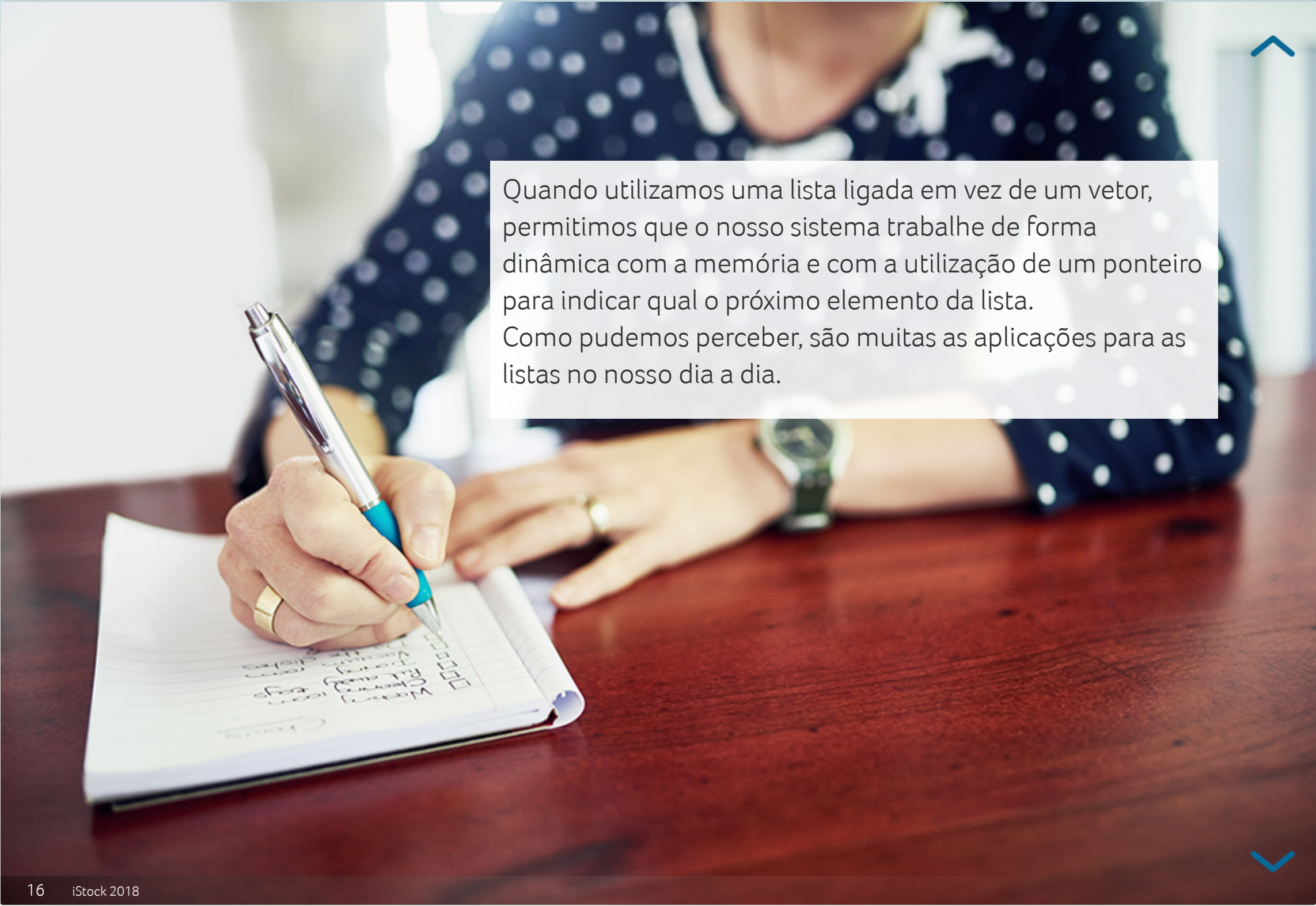
Aplicação de listas ligadas no nosso dia a dia

Explore a galeria e conheça aplicações comuns de listas no cotidiano.

Exemplo: criação de uma lista de compra de supermercado.
Ao anotar tudo que você precisa comprar, você automaticamente está criando uma lista, que pode ter produtos alocados ou removidos conforme o andamento da compra.

```
Dados *inicia_listaMerc(char *prod, int numpro) {  
    Dados *novo;  
    novo = (Dados *)malloc(sizeof(Dados));  
    novo -> prod = (char *)malloc(strlen(prod) + 1);  
    strncpy (novo -> prod, prod, strlen(prod) + 1);  
    novo -> numpro = numpro;  
    novo -> proximo = NULL;  
    return novo;  
}
```

Fonte: elaborado pelo autor (2017).

A person wearing a dark blue polka-dot shirt is sitting at a wooden desk, writing in a spiral-bound notebook. They are holding a silver and blue pen. The notebook has a checklist with several items, some of which are checked off. The person is also wearing a gold ring and a green watch. The background is slightly blurred, showing a white wall and a window.

Quando utilizamos uma lista ligada em vez de um vetor, permitimos que o nosso sistema trabalhe de forma dinâmica com a memória e com a utilização de um ponteiro para indicar qual o próximo elemento da lista. Como pudemos perceber, são muitas as aplicações para as listas no nosso dia a dia.

Você já conhece o Saber?

Aqui você tem na palma da sua mão a **biblioteca digital** para sua **formação profissional**.

Estude no celular, tablet ou PC em qualquer hora e lugar sem pagar mais nada por isso.

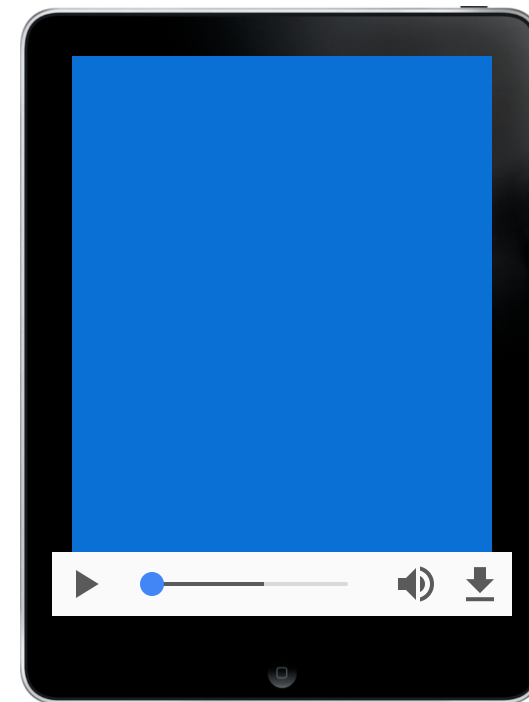
Mais de 450 livros com interatividade, vídeos, animações e jogos para você.



Android:
<https://goo.gl/yAL2Mv>



iPhone e iPad - IOS:
<https://goo.gl/OFWqcq>





Bons estudos!