

Apresentação

Relacionar itens em listas é uma atividade comum no dia a dia de muitas pessoas. Não raras vezes, criamos listas para nos auxiliar em diversas atividades. Montamos uma lista de compras de presentes de Natal, construímos uma relação de tarefas a serem executadas ou até definimos uma lista de presentes de casamento. **Na estrutura de dados, as listas apresentam formas adequadas para automatizar processos do mundo real**, de acordo com a necessidade ou o objetivo da situação, sem deixar de atender às características que definem os requisitos de manipulação de listas.

Nesta Unidade de Aprendizagem, **você vai estudar sobre TAD do tipo lista, bem como descrever e construir listas estáticas.**

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Reconhecer um TAD do tipo lista.
- Descrever os elementos de uma lista.
- Compor listas estáticas.

Desafio

Conhecer e implementar os métodos de acesso a listas que armazenam estruturas heterogêneas, como TADs, são atividades básicas da disciplina Estrutura de dados. As listas, de um modo geral, permitem o armazenamento e a manipulação de elementos independentemente da ordem em que foram inseridos, garantindo o requisito básico de serem sequenciais e não conterem elementos nulos.

Nesse contexto, este Desafio propõe a implementação de uma relação que armazene a lista de chamada de uma determinada disciplina, contendo o nome do aluno, a nota e o status da sua matrícula (ativa ou cancelada).

São métodos que devem ser implementados neste Desafio:

- Incluir e excluir alunos em qualquer ordem até o limite de 40;
- Ordenar a lista por nota decrescente;
- Apresentar a relação total de alunos e de alunos que estejam com a matrícula ativa.

Apresente as chamadas dos métodos de acesso à lista que demonstrem a utilização das funções implementadas. Utilize o seguinte TAD como modelo.

```
typedef struct {  
    char nome[20];  
    float nota;  
    char status;  
} Aluno;
```

Infográfico

Listas são relações de itens que fazem parte da rotina de qualquer pessoa. Podem ser encontradas nas mais diversas situações, como na hora de fazer compras no supermercado. Em Estrutura de dados, para uma relação ser considerada uma lista, precisa cumprir alguns requisitos, como permitir a inclusão e exclusão em qualquer posição. Além disso, a cada movimentação de elementos, deve reacomodar os demais itens para evitar espaços vazios, uma vez que uma lista deve ser sequencial e não pode conter elementos nulos.

Veja no Infográfico a apresentação de uma lista de supermercado e o seu comportamento quando se excluem ou incluem itens da relação.

LISTA DE COMPRAS DO SUPERMERCADO.



LISTA:

é uma relação de elementos dispostos sequencialmente que permite a inclusão e a exclusão de itens de qualquer posição, sem deixar elementos nulos entre os itens.

EXCLUSÃO DE ITEM:

qualquer item pode ser excluído da lista. Se a exclusão não ocorrer no final, os itens posteriores devem ser reacomodados na lista.

INCLUSÃO DE ITEM:

um item pode ser incluído em qualquer posição da lista. Se a inclusão não ocorrer no final, os itens posteriores devem ser repositionados na lista.

Conteúdo do Livro

Em Estrutura de dados, as listas apresentam diversas formas de manipular uma relação de elementos que representam alguma questão do mundo real. Uma das formas de representação é a lista estática, que permite criar arrays com capacidade de armazenar Tipos Abstratos de Dados (TAD) como elementos organizados em sequência.

As listas podem ser manipuladas por meio de métodos específicos, que podem ser implementados de acordo com a necessidade ou funcionalidade da aplicação, como inclusão, exclusão e modificação de itens, pesquisa e impressão, entre outros.

No capítulo Listas, do livro *Estrutura de dados*, base teórica desta Unidade de Aprendizagem, você vai estudar sobre TAD do tipo lista, bem como descrever e construir listas estáticas.

Boa leitura.

ESTRUTURA DE DADOS



A photograph showing four people working together at a table. A woman with curly hair is leaning over, looking at a tablet. A man is pointing at a document. Another person's hands are visible on the right. The table is covered with papers, a tablet, and a coffee cup. The background is blurred, suggesting an office or study environment. The entire image is overlaid with a pattern of overlapping hexagons in various colors (orange, purple, brown, blue, grey) of different sizes.

Adriana de Souza
Vettorazzo

Listas

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer um TAD do tipo lista.
- Descrever os elementos de uma lista.
- Compor listas estáticas.

Introdução

Relacionar itens em listas é uma atividade comum no dia a dia de muitas pessoas, seja para auxiliá-las em diversas atividades, como comprar presentes de natal, construir uma relação de tarefas a serem executadas ou até para definir os presentes de casamento. Na estrutura de dados, as listas apresentam formas adequadas para automatizar processos do mundo real, de acordo com a necessidade ou o objetivo da situação, sem deixar de atender às características que definem os requisitos de sua manipulação.

Neste capítulo, você estudará sobre o tipo abstrato de dados (TAD) do tipo lista, descreverá e construirá listas estáticas.

Tipo abstrato de dados do tipo lista

Um TAD é um dado definido com base nos tipos primitivos especificados pelas linguagens de programação, como *int*, *float* e *char*, entre outros. Com ele, é possível criar estruturas compostas de elementos de diversos tipos que podem ser tratadas de forma agrupada (CELES; CERQUEIRA; RANGEL, 2004).

Na linguagem C, uma estrutura TAD pode ser criada por meio da instrução *struct*.

```
typedef struct {  
    char nome[20];  
    float nota;  
    int turma;  
} Aluno;
```

Nesse exemplo, a estrutura **Aluno** é um TAD composto de três elementos diferentes: nome, como uma cadeia de caracteres; nota, como um número real; e turma, como um número inteiro. Desse modo, pode-se manipular os elementos da estrutura de forma individual ou agrupada, conforme você verá a seguir:

```
Aluno aluno;  
Strcpy(aluno.nome, "Mauricio";  
aluno.nota= 9.5;  
aluno.turma= 1;  
Imprimir(&aluno);
```

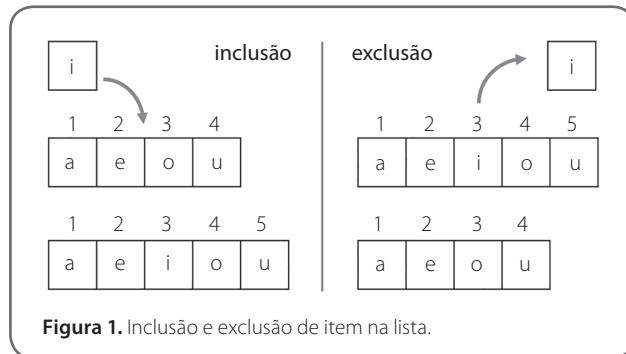
Já uma lista é um conjunto de dados que estão relacionados em determinada ordem linear. Cada item dela representa um nó da cadeia e pode conter elementos de apenas um tipo de dados primitivo ou de um tipo composto, como um TAD (EDELWEISS; GALANTE, 2009).



Fique atento

De modo geral, as listas são amplamente utilizadas em sistemas computacionais, nos quais diversas atividades do mundo real podem ser representadas por elas, como os itens de um carrinho de compras de um site *e-commerce*, a relação de alunos aprovados no vestibular de determinada universidade etc.

Os elementos de uma lista estão dispostos em uma ordem sequencial, em posições consecutivas, nas quais não pode haver qualquer elemento nulo. Assim, elementos inseridos ou excluídos da lista causarão deslocamento dos outros itens para reorganizá-la (MATTOS; LORENZI; CARVALHO, 2007). Na Figura 1, você verá como é a inclusão e a exclusão de um elemento na lista.



Na Figura 1, a inclusão do elemento **i**, entre as posições 2 e 3, causa deslocamento à direita das posições 3 e 4, que passam a ser 4 e 5, respectivamente. Já na exclusão, a sua retirada da posição 3 causa deslocamento à esquerda das posições 4 e 5, que passam a ser 3 e 4, respectivamente.



Fique atento

Uma lista estática do tipo TAD é determinada por uma quantidade limitada de itens, geralmente implementada por meio de um *array* de elementos de uma estrutura abstrata. Apesar de receber e perder itens, seu tamanho máximo é definido na sua criação, por isso, se caracteriza como estática.

Você conhecerá a seguir as vantagens, as desvantagens e a indicação de listas estáticas (FORBELLONE, 2005).

- **Vantagens:** o acesso direto aos elementos do *array* por meio de seu índice e o tempo constante a cada acesso direto, independentemente da quantidade de itens da lista.
- **Desvantagens:** o custo de processamento de inserção/exclusão ao precisar reposicionar os elementos da lista, e o tamanho máximo predefinido no momento de sua criação.
- **Indicação:** listas estáticas são recomendadas para listas pequenas, quando é necessário limitar seu tamanho máximo, como a escalação de um time titular de futebol, e sempre que a inclusão e a exclusão de itens puderem ser realizadas preferencialmente na última posição.

Uma lista estática do tipo TAD pode ser composta de dois elementos. O primeiro é conhecido como **dado** e define um *array* da estrutura, por exemplo, **dado[MAX]** que contém o nome, a nota e a turma de **Aluno**. Já o segundo elemento, nomeado de **n**, é um contador que armazena o total de itens da lista em tempo de execução.

```
#define MAX 5
typedef struct {
    Aluno dado[MAX];
    int n;
} ListaAlunos;
```

Você verá um exemplo de elementos TAD em uma lista estática na Figura 2.

	1	2	3	4	5
Nome	Paula	Pedro	Ana	Maria	João
Nota	9.1	8.9	8.7	9.3	9.8
Turma	1	2	1	3	3

Figura 2. Exemplo de elementos TAD em uma lista estática.

Uma vez que a estrutura da lista é definida e declarada como um *array* de elementos do tipo TAD, é necessário implementar a sua interface de acesso, que são as funções que permitem manipulá-la, como incluir e excluir itens, verificar se está vazia ou cheia, imprimir os seus elementos etc.

Interface de uma lista estática

Uma característica importante da lista estática é a separação entre a sua estrutura de dados e a implementação dos métodos que fazem a sua interface de acesso. Isso significa que um programador pode usar os seus métodos sem conhecer os detalhes de sua implementação (MATTOS; LORENZI; CARVALHO, 2007).

Não há uma definição de quais são exatamente os métodos de acesso de uma lista estática, porque eles podem ser definidos com base nas funcionali-

dades e necessidades da aplicação que vai utilizá-la, sendo as mais conhecidas (CELES; CERQUEIRA; RANGEL, 2004):

- criar uma lista vazia;
- identificar se a lista está vazia;
- identificar se a lista está cheia;
- excluir um item da lista;
- acessar uma posição da lista;
- inserir um item em uma posição da lista;
- inserir um item no final da lista;
- alterar um item da lista;
- identificar o número de itens da lista;
- concatenar duas listas;
- imprimir os elementos da lista;
- dividir a lista em duas ou mais;
- classificar a lista;
- inverter a ordem dos itens da lista.



Fique atento

Uma lista estática é mais genérica que filas e pilhas, porque não existe uma definição de ordem de entrada ou retirada de elementos. Isso significa que esses itens podem ser inseridos e excluídos de qualquer posição, desde que ela seja sequencial e não tenha elementos nulos (EDELWEISS; GALANTE, 2009).

Principais funções de acesso a listas estáticas

Criação de uma lista vazia

Uma lista pode ser criada/inicializada quando o seu contador for definido como zero, sem elementos, estando vazia e pronta para receber seus primeiros itens. O algoritmo a seguir apresenta o pseudocódigo dessa função.

```
Função criar (lista)
  Início
    lista.n= 0
  Fim Função
```

Identificação de uma lista vazia

Uma lista está vazia quando o seu contador de elementos for igual a zero. O pseudocódigo que ilustra o algoritmo dessa função será mostrado a seguir.

```
Função estaVazia (lista)
    Início
        Se lista.n Igual 0 Então
            Retorne Verdadeiro
        Fim Se
    Fim Função
```

Identificação de uma lista cheia

Verificar se a lista está cheia é uma das funções mais importantes, uma vez que ela não pode permitir mais elementos que a quantidade máxima prevista. Para identificar isso, compara-se o seu contador de itens com a constante que define o número máximo de elementos, conforme é descrito neste algoritmo:

```
Função estaCheia (lista)
    Início
        Se lista.n Igual MAX Então
            Retorne Verdadeiro
        Fim Se
    Fim Função
```

Identificação do número de itens da lista

O número de itens de uma lista equivale ao seu contador de elementos, que é atualizado a cada inserção ou exclusão. O algoritmo a seguir apresenta o pseudocódigo dessa função.

```
Função tamanho (lista)
    Início
        Retorne lista.n
    Fim Função
```

Inserção de item na lista

A inclusão de um item na lista deve seguir as seguintes regras:

- verificar se ela está cheia;
- abrir um espaço para inserir o novo item e reacomodar os elementos para não deixar alguma posição vazia, exceto se for inclui-lo no final da lista;
- incrementar o seu contador de itens.

A função que insere um novo elemento na lista deve receber três parâmetros: a lista que irá recebê-lo, a posição em que ele será inserido e o valor que deverá ser armazenado.

```
Função inserir (lista, posição, dado)
  Início
    Se listaCheia (lista)
      Retorne Falso
    Fim Se

    Para (i= lista.n, i Menor ou Igual posição; i= i-1)
      lista.dado[i]= dado[i-1]
    Fim Para

    lista.dado[posição-1]= dado
    lista.n= lista.n + 1
    Retorne Verdadeiro
  Fim Função
```

Exclusão de item de uma posição da lista

Assim como a inclusão de itens, a exclusão deve seguir alguns critérios:

- verificar se ela está vazia;
- fechar o espaço deixado pelo elemento excluído, reacomodando os demais itens, exceto se for exclui-lo do final da lista;
- decrementar o seu contador de itens.

A função que exclui um elemento da lista deve receber dois parâmetros: a lista que terá o item retirado e a sua posição.

```

Função remover (lista, posição)
Início
    Se listaVazia (lista)
        Retorne Falso
    Fim Se

    Para (i= posição; i Menor ou Igual lista.n-1; i= i + 1)
        lista.dado[i-1] = lista.dado[i]
    Fim Para

    lista.n= lista.n - 1
    Retorne Verdadeiro
Fim Função

```

Impressão dos elementos de uma lista

Para imprimir a relação de itens de uma lista, o algoritmo deve percorrer todos os elementos do *array*, partindo do início até o seu contador de itens, que indica o seu número total.

```

Função imprimir (lista)
Início
    Para (i= 1; i Menor ou Igual lista.n; i= i + 1)
        Escreva lista.dado[i]
    Fim Para
Fim Função

```

Construção de listas estáticas

Você conhecerá a implementação, em linguagem C, dos principais métodos de acesso da lista estática baseada em um TAD, que foram ilustrados em pseudocódigo na seção anterior. Para isso, foram criados três arquivos.

- lista.h: arquivo de cabeçalho que contém os protótipos das funções e a definição da estrutura TAD da lista.
- lista.c: arquivo que contém a implementação das funções.
- hello.c: arquivo que contém o método *main* e declara uma lista TAD para chamar os métodos de acesso.

A estrutura TAD que será apresentada corresponde aos dados de um aluno e contém três atributos: nome, nota e turma. Ela será armazenada em um *array* de cinco elementos, que equivale aos itens da lista utilizada.

Arquivo lista.h

Esse arquivo define o número máximo de elementos da lista, a estrutura TAD do aluno, o *array* que vai armazenar os dados desses alunos em cada posição e os protótipos das funções de acesso a ela.

```
1 #define MAX 5
2 typedef struct {
3     char nome[20];
4     float nota;
5     int turma;
6 } Aluno;
7
8 typedef struct {
9     Aluno dado[MAX];
10    int n;
11 } ListaAlunos;
12
13 //cria uma lista de alunos
14 void criar(ListaAlunos *);
15
16 //retorna 1 se está vazia e 0 se não estiver
17 int estaVazia(ListaAlunos);
18
19 //função estaCheia
20 //retorna 1 se está cheia e 0 se não estiver
21 int estaCheia(ListaAlunos);
22
23 //insere um aluno em determinada posição da lista.
24 //empurra os elementos subsequentes para inserir.
25 //retorna 1 se sucesso e 0 se falhar
26 int inserir (ListaAlunos *, int, Aluno);
27
28 //imprime a lista de alunos.
29 int imprimir (ListaAlunos *);
30
31 //retorna o tamanho da lista.
32 int tamanho(ListaAlunos);
33
34 //remove um aluno de determinada posição da lista.
```

Arquivo lista.c

Esse arquivo contém a implementação das funções de acesso à lista. Ele requer a inclusão do arquivo lista.h que, por sua vez, inclui os protótipos das funções, bem como a sua estrutura de definição.

```
1 #include <stdio.h>
2 #include <string.h>
3 #include "lista.h"
4
5 void criar(ListaAlunos *L){
6     L->n= 0;
7 }
8
9 int estaVazia(ListaAlunos L){
10    return (L.n == 0);
11 }
12
13 int estaCheia(ListaAlunos L){
14    return (L.n == MAX);
15 }
16
17 int inserir (ListaAlunos *L, int pos, Aluno aluno) {
18     if ((L->n == MAX) || (pos > L->n + 1))
19         return 0;
20
21     int i;
22     for (i= L->n; i >= pos; i--)
23         L->dado[i]= L->dado[i-1];
24
25     L->dado[pos-1]= aluno;
26     (L->n)++;
27     return 1;
28 }
```

```
29
30 int tamanho(ListaAlunos L){
31     return(L.n);
32 }
33
34 int imprimir (ListaAlunos *L){
35     int i;
36     printf("%-4s %-10s %-6s %-5s \n",
37            "Pos", "Nome", "Nota", "Turma");
38     int total= L->n;
39     for (i= 0; i < total; i++)
40         printf("%-4d %-10s %-6.1f %-4d \n",
41                i+1, L->dado[i].nome, L->dado[i].nota, L->dado[i].turma);
42     printf("\n");
43 }
44
45 int remover (ListaAlunos *L, int pos){
46     int i;
47     if ((pos > L->n) || (pos <= 0))
48         return 0;
49
50     for (i= pos; i <= (L->n)-1; i++)
51         L->dado[i-1]= L->dado[i];
52
53     (L->n)--;
54     return 1;
55 }
```

Arquivo hello.c

Esse arquivo contém o módulo principal do exemplo, porque nele é realizada a declaração do *array* da estrutura TAD **Aluno** e as chamadas das respectivas funções de acesso à lista. Para acessar os métodos da lista, bem como a sua estrutura, é preciso realizar a inclusão do arquivo lista.c.

Já para testar a inclusão de elementos na lista, deve-se criar uma função, chamada “**instanci**”, que irá preparar os dados de entrada, tendo como parâmetros as informações do aluno: nome, nota e turma.

Nesse exemplo, você acompanhará a criação de uma lista de cinco posições que armazenam a estrutura **Aluno** e a realização de algumas operações por meio dos seus métodos, como cria-la, inserir e remover elementos, verificar se está cheia e vazia, mostrar a quantidade de itens e imprimir a relação completa de seus alunos.

```
1 #include "lista.cpp"
2
3 void instancia(Aluno *, char *, float, int);
4
5 void instancia(Aluno *aluno, char *nome, float nota, int turma){
6     strcpy(aluno->nome, nome);
7     aluno->nota= nota;
8     aluno->turma= turma;
9 }
10
11 main(){
12     ListaAlunos lista;
13     Aluno aluno;
14     int n;
15
16     criar(&lista);
17     printf("Lista criada! \n");
18
19     n= estaVazia(lista);
20     printf("Lista esta vazia? %s \n", n==0 ? "Nao" : "sim");
21
22     n= estaCheia(lista);
23     printf("Lista esta cheia? %s \n\n", n==0 ? "Nao" : "sim");
24
25     instancia(&aluno, "Pedro", 8.9, 2);
26     inserir(&lista, 1, aluno);
27     printf("Pedro adicionado na posicao 1. \n");
28
29     instancia(&aluno, "Ana", 8.7, 1);
30     inserir(&lista, 2, aluno);
31     printf("Ana adicionada na posicao 2. \n");
32
33     instancia(&aluno, "Maria", 9.3, 3);
34     inserir(&lista, 3, aluno);
```

```
35     printf("Maria adicionada na posicao 3. \n\n");
36
37     imprimir(&lista);
38
39     remover(&lista, 1);
40     printf("Pedro removido da posicao 1. \n");
41
42     imprimir(&lista);
43
44     instancia(&aluno, "Paula", 9.1, 1);
45     inserir(&lista, 1, aluno);
46     printf("Paula adicionada na posicao 1. \n");
47
48     imprimir(&lista);
49
50     instancia(&aluno, "Pedro", 8.9, 2);
51     inserir(&lista, 2, aluno);
52     printf("Pedro adicionado na posicao 2. \n");
53
54     instancia(&aluno, "Joao", 9.8, 3);
55     inserir(&lista, 5, aluno);
56     printf("Joao adicionado na posicao 5. \n\n");
57
58     imprimir(&lista);
59
60     n= estaCheia(lista);
61     printf("Lista esta cheia? %s \n", n==0 ? "Nao" : "sim");
62
63     n= tamanho(lista);
64     printf("Tamanho atual da lista: %d \n\n", n);
65 }
```

O resultado da execução desse programa está ilustrado na Figura 3. Inicialmente, a lista foi criada e verificou-se se ela estava vazia e cheia (linhas 16 a 23). Na sequência, três alunos foram adicionados: Pedro, Ana e Maria, nessa mesma ordem, e a relação foi impressa (linhas 25 a 37).

Em seguida, o aluno Pedro foi removido da primeira posição; e a lista para ilustrar a retirada, impressa (linhas 39 a 42). Paula foi adicionada na primeira posição, Pedro na segunda e João na quinta; e a lista, impressa novamente (linhas 44 a 58). Por fim, verificou-se se a lista estava cheia, e a sua quantidade de elementos foi mostrada (linhas 60 a 64).

```
D:\M\Sagah\Disciplinas\Estrutura de Dados\UA13_Listas_estaticas\arquivos\hello.exe
Lista criada!
Lista esta vazia? sim
Lista esta cheia? Nao

Pedro adicionado na posicao 1.
Ana adicionada na posicao 2.
Maria adicionada na posicao 3.

Pos  Nome      Nota   Turma
1   Pedro      8.9    2
2   Ana        8.7    1
3   Maria      9.3    3

Pedro removido da posicao 1.
Pos  Nome      Nota   Turma
1   Ana        8.7    1
2   Maria      9.3    3

Paula adicionada na posicao 1.
Pos  Nome      Nota   Turma
1   Paula     9.1    1
2   Ana        8.7    1
3   Maria      9.3    3

Pedro adicionado na posicao 2.
Joao adicionado na posicao 5.

Pos  Nome      Nota   Turma
1   Paula     9.1    1
2   Pedro      8.9    2
3   Ana        8.7    1
4   Maria      9.3    3
5   Joao       9.8    3

Lista esta cheia? sim
Tamanho atual da lista: 5
```

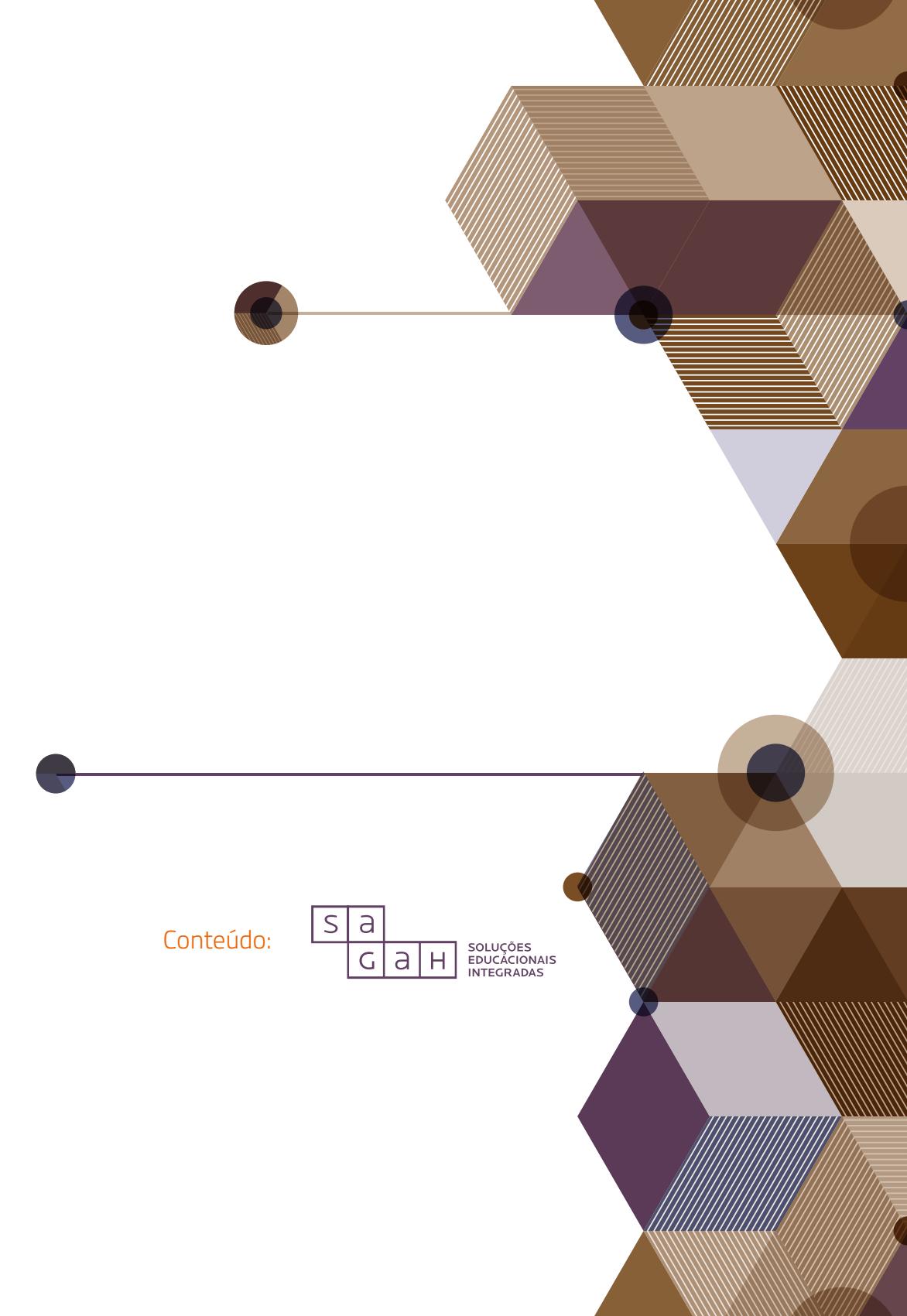
Figura 3. Resultado da execução do programa de exemplo.



Referências

- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estrutura de dados*. São Paulo: Campus, 2004.
- EDELWEISS, N.; GALANTE, R. *Estrutura de dados*. Porto Alegre: Bookman, 2009. (Série Livros Didáticos Informática UFRGS, v. 18).
- FORBELLONE, A. L. *Lógica de programação: a construção de algoritmos e estruturas de dados*. 3. ed. São Paulo: Pearson, 2005.
- MATTOS, P.; LORENZI, F.; CARVALHO, T. *Estruturas de dados*. São Paulo: Thomson, 2007.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.



Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Dica do Professor

Listas são amplamente utilizadas em **sistemas computacionais**. Frequentemente, programadores se deparam com a necessidade de criar e manipular **listas estáticas**, seja para apresentar uma simples relação de itens, seja para manipular estruturas complexas, compostas por **Tipos Abstratos de Dados**.

Nesta Dica do Professor, vamos tratar de **TAD do tipo lista e listas estáticas**.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

- 1) Baseando-se no conceito de lista estática, marque a alternativa correta em relação à inclusão de elementos.**
 - A) A inclusão de elementos ocorre no final da lista.
 - B) A inclusão de elementos ocorre no início da lista.
 - C) A inclusão de elementos pode ocorrer em qualquer posição da lista.
 - D) Não existe um limite para o número de elementos a serem incluídos.
 - E) A inclusão deve ocorrer em uma posição que esteja vazia.
- 2) Em relação às listas estáticas, leia as alternativas a seguir e indique a correta.**
 - A) O último elemento pode ser nulo para acomodar o próximo item a ser incluído.
 - B) Os itens estão dispostos em uma ordem sequencial, sem conter elementos nulos.
 - C) São indicadas para listas de qualquer tamanho.
 - D) O tamanho pode ser redimensionado em tempo de execução do programa.
 - E) Uma desvantagem é o alto custo computacional de acesso direto a determinada posição.
- 3) Marque a alternativa correta em relação à interface de uma lista estática.**
 - A) Nela, há as funções nativas da linguagem de programação C.
 - B) Os detalhes da implementação dos métodos de acesso devem estar disponíveis para o programador.
 - C) Não existe uma definição de quais métodos devem ser implementados, embora alguns sejam esperados.
 - D) Os métodos de acesso têm o mesmo comportamento de outras estruturas semelhantes, como pilhas e filas.

E) Recomenda-se unir a estrutura de uma lista estática com a implementação de seus métodos de acesso.

4) Analise o seguinte pseudocódigo baseado na linguagem C e marque a alternativa que representa o significado da função xxxx.

```
typedef struct {  
    int dado[50]; //array de itens da lista  
    int n; //total de elementos da lista  
} Lista;  
  
int xxxx (Lista *L, int pos){  
    for (int i= pos; i <= (L->n)-1; i++)  
        L->dado[i-1]= L->dado[i];  
    (L->n)--;  
}
```

A) Remove o primeiro item da lista e reposiciona os demais.

B) Adiciona um item na lista na posição "pos" e reposiciona os demais.

C) Remove o último item da lista e reposiciona os demais.

D) Adiciona um item na primeira posição da lista e reposiciona os demais.

E) Remove o item da lista de posição igual a "pos" e reposiciona os demais.

5) Analise o seguinte código baseado na linguagem C indique a alternativa que o define.

```
typedef struct {  
    unsigned long CPF;
```

```
char Ativo;  
}  
Documento;  
  
typedef struct {  
    Documento Cadastro[1000];  
    int n;  
}  
Pessoas;
```

Pessoas lista;

Documento doc;

- A) Define uma lista para armazenar 1000 elementos do tipo Documento.
- B) Define uma lista para armazenar 1000 elementos do tipo Pessoas.
- C) Define uma lista para armazenar 1000 elementos do tipo Cadastro.
- D) Define uma lista para armazenar 1000 elementos do tipo CPF.
- E) Define uma lista para armazenar 1000 elementos do tipo Ativo.

Na prática

Nesta Unidade de Aprendizagem, vamos apresentar um TAD do tipo lista, composto dos seguintes atributos: nome, altura, peso e adimplente, que compreendem o cadastro de alunos de uma academia de ginástica.

Para administrar essa estrutura, vamos criar uma lista estática que pode receber até 50 posições, na qual se implementarão os seguintes métodos de acesso: acrescentar e remover elementos, imprimir, ordenar crescentemente por nome, criar sublista de adimplentes e calcular a média de altura dos alunos.

A definição da estrutura Aluno e das listas (geral e adimplentes) está descrita a seguir

```
#define MAX 50

typedef struct {

    char nome[20];

    float altura;

    float peso;

    char adimplente;

} Aluno;

typedef struct {

    Aluno dado[MAX];

    int n= 0;

} ListaAcademia;
```

```
ListaAcademia lista;  
  
ListaAcademia listaAdimilentes;  
  
Aluno aluno;
```

Com essa definição, é possível criar diversas listas baseadas no TAD *Aluno*. Assim, neste Na Prática, vamos criar duas listas: uma lista geral, nomeada simplesmente de *lista*, e uma lista de alunos adimplentes, nomeada de *listaAdimilentes*.

Implementamos as funções básicas de acesso às listas, conforme descrito nos seguintes protótipos de funções.

```
void instancia(Aluno *, char *, float, float, char);  
  
void ordenar(ListaAcademia *);  
  
int acrescentar (ListaAcademia *, Aluno);  
  
int remover (ListaAcademia *, int);  
  
int imprimir (ListaAcademia *);  
  
void criarAdimilentes(ListaAcademia *, ListaAcademia *);  
  
float mediaAltura(ListaAcademia *);
```

Destacam-se, nessa relação de métodos de acesso, as funções *ordenar* e *criarAdimilentes*, que realizam, respectivamente, a ordenação da lista por nome de aluno e a criação de uma sublista com base nas informações da lista geral.

Definimos a implementação da função *instancia* para facilitar a entrada de dados na estrutura por meio dos parâmetros que definem os dados do aluno, como *nome*, *altura*, *peso* e *adimilente*.

```
void instancia(Aluno *aluno, char *nome, float altura,  
float peso, char adimilente) {  
  
strcpy(aluno->nome, nome);
```

```
aluno->altura= altura;  
  
aluno->peso= peso;  
  
aluno->adimplente= adimplente;  
  
}
```

A função *ordenar* utiliza a técnica de ordenação conhecida por *Ordenação por Seleção*. Essa técnica é ideal para listas pequenas, como as listas estáticas apresentadas nesta prática. O algoritmo dessa função está descrito a seguir.

```
void ordenar(ListaAcademia *L) {  
  
    int i, j, min;  
  
    int n= L->n;  
  
    Aluno temp;  
  
    for (i= 0; i < n-1; i++) {  
  
        min= i;  
  
        for (j= i+1; j < n; j++) {  
  
            if (strcmp(L->dado[j].nome, L->dado[min].nome) < 0)  
                min= j;  
  
        }  
  
        temp= L->dado[min];  
  
        L->dado[min]= L->dado[i];  
  
        L->dado[i]= temp;  
  
    }  
  
}
```

Já o método *criarAdimplentes* faz uma varredura na lista geral e adiciona cada aluno adimplente na nova lista: *listaAdimplentes*. Isso faz o exemplo trabalhar com duas listagens, que podem ser manipuladas separadamente.

```
void criarAdimplentes(ListaAcademia *ListaGeral,
                      ListaAcademia *ListaAdimplentes) {
    int i;
    int n= ListaGeral->n;
    Aluno aluno;
    for (i= 0; i < n; i++) {
        if (ListaGeral->dado[i].adimplente == 'S') {
            aluno= ListaGeral->dado[i];
            acrescentar(ListaAdimplentes, aluno);
        }
    }
}
```

Para testar as implementações, declaramos o método principal e realizamos as chamadas das funções.

Primeiramente, criamos as instâncias para colocar na lista geral os dados dos alunos.

```
instancia(&aluno, "Paula", 1.60, 63.8, 'S');
acrescentar(&lista, aluno);

instancia(&aluno, "Pedro", 1.80, 81.5, 'S');
```

```
acrescentar(&lista, aluno);  
  
instancia(&aluno, "Ana", 1.52, 49.9, 'N');  
  
acrescentar(&lista, aluno);  
  
instancia(&aluno, "Maria", 1.69, 58.0, 'S');  
  
acrescentar(&lista, aluno);  
  
instancia(&aluno, "Joao", 1.75, 90.1, 'N');  
  
acrescentar(&lista, aluno);
```

Na sequência, realizamos a ordenação da lista pela chamada da função *ordenar* e criamos a sublista dos alunos adimplentes, pela chamada da função *criarAdimplentes*. Essa função recebe dois parâmetros, a lista geral e a nova lista que será populada.

```
ordenar(&lista);  
  
criarAdimplentes(&lista, &listaAdimplentes);
```

Por fim, removemos um aluno da lista geral por ter solicitado o cancelamento de sua matrícula da academia e imprimimos a média de altura dos alunos de ambas as listas.

```
remover(&lista, 2);  
  
m= mediaAltura(&lista);  
  
m= mediaAltura(&listaAdimplentes);
```

O resultado da execução do programa está ilustrado pela seguinte figura:

Paula adicionada.
Pedro adicionado.
Ana adicionada.
Maria adicionada.
Joao adicionado.

Lista geral de alunos da academia:

Pos	Aluno	Altura	Peso	Adimplente
1	Paula	1.60	63.8	S
2	Pedro	1.80	81.5	S
3	Ana	1.52	49.9	N
4	Maria	1.69	58.0	S
5	Joao	1.75	90.1	N

Lista geral ordenada por nome:

Pos	Aluno	Altura	Peso	Adimplente
1	Ana	1.52	49.9	N
2	Joao	1.75	90.1	N
3	Maria	1.69	58.0	S
4	Paula	1.60	63.8	S
5	Pedro	1.80	81.5	S

Sublista adimplentes que podem ganhar desconto:

Pos	Aluno	Altura	Peso	Adimplente
1	Maria	1.69	58.0	S
2	Paula	1.60	63.8	S
3	Pedro	1.80	81.5	S

Joao removido da lista geral.

Lista geral de alunos da academia:

Pos	Aluno	Altura	Peso	Adimplente
1	Ana	1.52	49.9	N
2	Maria	1.69	58.0	S
3	Paula	1.60	63.8	S
4	Pedro	1.80	81.5	S

A media de altura da lista geral eh: 1.65

A media de altura da lista adimplentes eh: 1.70



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

O código completo desse programa é apresentado a seguir:

```
#include <stdio.h>

#include <string.h>

#define MAX 50

typedef struct {

    char nome[20];

    float altura;

    float peso;

    char adimplente;

} Aluno;

typedef struct {

    Aluno dado[MAX];

    int n= 0;

} ListaAcademia;

ListaAcademia lista;

ListaAcademia listaAdimplentes;

Aluno aluno;
```

```
//Protótipos das funções

void instancia(Aluno *, char *, float, float, char);

void ordenar(ListaAcademia *);

int acrescentar (ListaAcademia *, Aluno);

int remover (ListaAcademia *, int);

int imprimir (ListaAcademia *);

void criarAdimplentes(ListaAcademia *, ListaAcademia *);

float mediaAltura(ListaAcademia *);

//Implementação das funções de acesso à lista

void instancia(Aluno *aluno, char *nome, float altura,
float peso, char adimplente) {

strcpy(aluno->nome, nome);

aluno->altura= altura;

aluno->peso= peso;

aluno->adimplente= adimplente;

}

int acrescentar (ListaAcademia *L, Aluno aluno) {

int total= L->n;

if (total >= MAX)

return 0;
```

```
L->dado[total]= aluno;
(L->n)++;
return 1;
}

int remover (ListaAcademia *L, int pos){
int i;
if ((pos > L->n) || (pos <= 0))
return 0;

for (i= pos; i <= (L->n)-1; i++)
L->dado[i-1]= L->dado[i];

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

int imprimir (ListaAcademia *L) {
int i;
printf("%-3s %-10s %-6s %-5s %-10s n", "Pos",
"Aluno", "Altura", "Peso", "Adimplente");
printf("%-3s %-10s %-6s %-5s %-10s n", "----", "-----",
"-----", "----", "-----");
```

```
int total= L->n;

for (i= 0; i < total; i++)

printf("%-3d %-10s %-6.2f %-5.1f %-10c n",
i+1, L->dado[i].nome, L->dado[i].altura,
L->dado[i].peso, L->dado[i].adimplente);

printf("n");

}
```

```
void ordenar(ListaAcademia *L) {

int i, j, min;

int n= L->n;

Aluno temp;

for (i= 0; i < n-1; i++) {

min= i;

for (j= i+1; j < n; j++) {

if (strcmp(L->dado[j].nome,
L->dado[min].nome) < 0)

min= j;

temp= L->dado[min];

L->dado[min]= L->dado[i];

L->dado[i]= temp;

}
```

```
}
```

```
}
```

```
void criarAdimplentes(ListaAcademia *ListaGeral,
ListaAcademia *ListaAdimplentes) {
int i;
int n= ListaGeral->n;
Aluno aluno;
for (i= 0; i < n; i++) {
if (ListaGeral->dado[i].adimplente == 'S') {
aluno= ListaGeral->dado[i];
acrescentar(ListaAdimplentes, aluno);
}
}
}
```

```
float mediaAltura(ListaAcademia *L) {
int i;
int n= L->n;
float somaAltura= 0;
for (i= 0; i < n; i++) {
somaAltura+= L->dado[i].altura;
}
return (float) somaAltura / n;
```

}

```
main() {  
    float m;  
  
    instancia(&aluno, "Paula", 1.60, 63.8, 'S');  
    acrescentar(&lista, aluno);  
    printf("Paula adicionada. n");  
  
    instancia(&aluno, "Pedro", 1.80, 81.5, 'S');  
    acrescentar(&lista, aluno);  
    printf("Pedro adicionado. n");  
  
    instancia(&aluno, "Ana", 1.52, 49.9, 'N');  
    acrescentar(&lista, aluno);  
    printf("Ana adicionada. n");  
  
    instancia(&aluno, "Maria", 1.69, 58.0, 'S');  
    acrescentar(&lista, aluno);  
    printf("Maria adicionada. n");  
  
    instancia(&aluno, "Joao", 1.75, 90.1, 'N');  
    acrescentar(&lista, aluno);  
    printf("Joao adicionado. nn");
```

```
printf("Lista geral de alunos da academia: n");
imprimir(&lista);

ordenar(&lista);

printf("Lista geral ordenada por nome: n");
imprimir(&lista);

criarAdimplentes(&lista, &listaAdimplentes);

printf("Sublista adimplentes que podem ganhar desconto: n");
imprimir(&listaAdimplentes);

remover(&lista, 2);

printf("Joao removido da lista geral. nn");

printf("Lista geral de alunos da academia: n");
imprimir(&lista);

m= mediaAltura(&lista);

printf("A media de altura da lista geral eh: %.2f n", m);

m= mediaAltura(&listaAdimplentes);

printf("A media de altura da lista adimplentes eh: %.2f n", m);
}
```

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Lista Estática Sequencial

Assista ao vídeo a seguir, que aborda o tema lista estática sequencial.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para acessar.

Estruturas de dados - Lista estática

Assista ao vídeo a seguir para saber mais sobre a implementação de métodos de uma lista estática.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para acessar.

Implementando Lista Estática

Assista ao vídeo a seguir para saber mais sobre como implementar uma lista estática sequencial.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para acessar.