

# QXD0010 – ESTRUTURA DE DADOS – 01A – 2025.2

[Página inicial](#)

▶

[Meus cursos](#)

▶

[QXD0010 – ESTRUTURA DE DADOS – 01A – 2025.2](#)

▶

[Tópico 9. Lista Simplesmente Encadeada](#)

▶

[\[linked list\] Implementar uma lista simplesmente encadeada - Parte 3](#)

☰ Descrição

📁 Enviar

📄 Editar

📁 Visualizar envios

## [linked list] Implementar uma lista simplesmente encadeada – Parte 3

📅 **Data de entrega:** domingo, 7 dez 2025, 23:59

📁 **Arquivos requeridos:** main.c, LinkedList.h, LinkedList.c (📄 Baixar)

📄 **Número máximo de arquivos:** 5

**Tipo de trabalho:** 👤 Trabalho individual



## Motivação (Listas Simplesmente Encadeadas)

Neste exercício, você deve completar a implementação da lista simplesmente encadeada com novas funções.

Os protótipos das funções que devem ser implementadas estão listados abaixo juntamente com um comentário explicando o que cada função deve fazer.

```
// Esta função troca o conteúdo das duas listas passadas como parâmetro.
void list_troca(Node **p1, Node **p2);

// Essa função recebe como entrada uma lista e
// coloca os elementos pares no início, e os elementos ímpares no fim da lista.
// Retorna a lista modificada. Não é permitido usar arrays nem vetores dinâmicos.
// Só é permitido usar ponteiro auxiliar (uma quantidade constante deles).
// Deve-se fazer com complexidade linear O(n).
Node* list_separa_pares_impares(Node *p);

// Essa função recebe como entrada uma lista encadeada com um número ímpar de elementos e
// Encontra o elemento central da lista e retorna um ponteiro para ele.
// Se a lista for vazia, retorna NULL.
// Por exemplo, se tivermos a lista [2,3,4,5,6], a função retorna um ponteiro para o 4.
Node* list_encontra_centro(Node *p);
```

**Observação 1:** As funções possuem restrições que devem ser obedecidas. Se as restrições não forem satisfeitas, haverá redução da nota.

**Observação 2:** O arquivo **main.c** e o arquivo **LinkedList.h** já foram codificados e foram incluídos nesta atividade.

Resta implementar apenas as funções listadas acima.

O programa principal em main.c lê comandos passados como entrada e manipula uma ou mais listas.

Os comandos aceitos pelo menu principal estão listados abaixo.

Comando	Significado
exit	sair do programa
create	cria uma nova lista vazia

troca p q	realiza a troca das listas p e q
show	mostra todas as listas na tela
separa p	modifica a lista p para que os pares fiquem à esquerda e ímpares à direita
find_center p	encontra o nó central da lista p e imprime seu valor na tela
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no início da lista k
addBack k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no final da lista k

Arquivos requeridos  
main.c

```
1 // ESTE ARQUIVO JA ESTA PRONTO, NAO MEXA NELE
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <stdbool.h>
6 #include "LinkedList.h"
7
8 // Vetor dinâmico simples
9 typedef struct {
10     Node **data;
11     size_t size;
12 } Vector;
13
14 // Inicializa vetor
15 void vector_init(Vector *v) {
16     v->data = NULL;
17     v->size = 0;
18 }
19
20 // Adiciona uma lista nova no final
21 void vector_push_back(Vector *v, Node *list) {
22     v->data = realloc(v->data, (v->size + 1) * sizeof(Node*));
23     v->data[v->size] = list;
24     v->size++;
25 }
26
27 // Libera todas as listas
28 void vector_clear(Vector *v) {
29     for(size_t i = 0; i < v->size; i++) {
30         list_free(v->data[i]);
31     }
32     free(v->data);
33     v->data = NULL;
34     v->size = 0;
35 }
36
37 int main() {
38     Vector listas;
39     vector_init(&listas);
40
41     char comando[300];
42
43     while (true) {
44
45         if(!fgets(comando, sizeof(comando), stdin))
46             break;
47
48         // Remove \n
49         comando[strcspn(comando, "\n")] = 0;
50
51         printf("$%s\n", comando);
52
53         char *token = strtok(comando, " ");
54
55         if(token == NULL)
56             continue;
57
58         // ----- exit -----
59         if(strcmp(token, "exit") == 0) {
60             vector_clear(&listas);
61             break;
62         }
63
64         // ----- create -----
65         else if(strcmp(token, "create") == 0) {
66             vector_push_back(&listas, list_create());
67         }
68
69         // ----- size l -----
70         else if(strcmp(token, "size") == 0) {
71             int l = atoi(strtok(NULL, " "));
72             printf("size list %d: %zu\n", l, list_size(listas.data[l]));
73         }
74
75         // ----- find_center l -----
76         else if(strcmp(token, "find_center") == 0) {
77             int l = atoi(strtok(NULL, " "));
78             Node *p = list_encontra_centro(listas.data[l]);
79             printf("elemento central: %d\n", list_pegav valor(p));
80         }
81
82         // ----- separa l -----
83         else if(strcmp(token, "separa") == 0) {
84             int l = atoi(strtok(NULL, " "));
85             listas.data[l] = list_separa_pares_impares(listas.data[l]);
86         }
87
88         // ----- troca p q -----
89         else if(strcmp(token, "troca") == 0) {
90             int p = atoi(strtok(NULL, " "));
91             int q = atoi(strtok(NULL, " "));
92             list_troca(&listas.data[p], &listas.data[q]);
93         }
94
95         // ----- addFront l a1 a2 ... -----
96         else if(strcmp(token, "addFront") == 0) {
97             int l = atoi(strtok(NULL, " "));
98             char *p = NULL;
99             while((p = strtok(NULL, " ")) != NULL) {
100                 int val = atoi(p);
101                 listas.data[l] = list_push_front(listas.data[l], val);
102             }
103         }
104
105         // ----- addBack l a1 a2 ... -----
106         else if(strcmp(token, "addBack") == 0) {
107             int l = atoi(strtok(NULL, " "));
108             char *p = NULL;
109             while((p = strtok(NULL, " ")) != NULL) {
110                 int val = atoi(p);
111                 listas.data[l] = list_push_back(listas.data[l], val);
112             }
113         }
114
115         // ----- show -----
116         else if(strcmp(token, "show") == 0) {
117             int l = atoi(strtok(NULL, " "));
118             list_show(listas.data[l]);
119         }
120     }
121 }
```

```
117         else if(strcmp(token, "show") == 0) {
118             for(size_t i = 0; i < listas.size; i++) {
119                 printf("lista %zu: ", i);
120                 list_print(listas.data[i]);
121             }
122         }
123
124         else {
125             printf("comando inexistente\n");
126         }
127     }
128
129     return 0;
130 }
```

LinkedList.h

```
1  // ESTE ARQUIVO JA ESTA PRONTO, NAO MEXA NELE
2  #ifndef LINKED_LIST_H
3  #define LINKED_LIST_H
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdbool.h>
7
8  typedef struct node Node;
9
10 // Função que cria uma lista vazia
11 // sem nó sentinela
12 Node *list_create(void);
13
14 // Insere um valor no início da lista
15 // e retorna a lista modificada.
16 Node *list_push_front(Node *list, int value);
17
18 // Insere um valor ao final da lista
19 // e retorna a lista modificada.
20 Node *list_push_back(Node *list, int value);
21
22 // Função que recebe um ponteiro para a cabeça da lista e
23 // imprime o conteúdo dos elementos na tela.
24 // Formato: [ a1 a2 a3 ... an ]
25 void list_print(Node *list);
26
27 // Função que libera todos os nós
28 // alocados dinamicamente
29 void list_free(Node *list);
30
31 // Função que imprime o tamanho da lista (número de nós)
32 size_t list_size(Node *list);
33
34 // Recebe como entrada um nó e devolve o valor dele.
35 int list_pegar_valor(Node *p);
36
37 // Esta função troca o conteúdo das duas listas passadas como parâmetro.
38 void list_troca(Node **p1, Node **p2);
39
40 // Essa função recebe como entrada uma lista e
41 // coloca os elementos pares no início, e os elementos ímpares no fim da lista.
42 // Retorna a lista modificada. Não é permitido usar arrays nem vetores dinâmicos.
43 // Só é permitido usar ponteiro auxiliar (uma quantidade constante deles).
44 // Deve-se fazer com complexidade linear O(n).
45 Node* list_separa_pares_impares(Node *p);
46
47 // Essa função recebe como entrada uma lista encadeada com um número ímpar de elementos e
48 // Encontra o elemento central da lista e retorna um ponteiro para ele.
49 // Se a lista for vazia, retorna NULL.
50 // Por exemplo, se tivermos a lista [2,3,4,5,6], a função retorna um ponteiro para o 4.
51 Node* list_encontra_centro(Node *p);
52
53 #endif
```

LinkedList.c

```
1  #include "LinkedList.h"
2
3  struct node {
4      int data;
5      struct node *next;
6  };
7
8
9  // Função que cria uma lista vazia
10 // sem nó sentinela
11 Node *list_create(void) {
12     return NULL;
13 }
14
15 // Insere um valor no início da lista
16 // e retorna a lista modificada.
17 Node *list_push_front(Node *list, int value) {
18     Node* novo = (Node*) malloc(sizeof(Node));
19     novo->data = value;
20     novo->next = list;
21     return novo;
22 }
23
24 // Insere um valor ao final da lista
25 // e retorna a lista modificada.
26 Node *list_push_back(Node *list, int value) {
27     Node *novo = (Node *)malloc(sizeof(Node));
28     novo->data = value;
29     novo->next = NULL;
30
31     if(list == NULL) {
32         return novo;
33     } else {
34         Node *q = list;
35         while(q->next != NULL) {
36             q = q->next;
37         }
38         q->next = novo;
39         return list;
40     }
41 }
42
43 // Função que recebe um ponteiro para a cabeça da lista e
44 // imprime o conteúdo dos elementos na tela.
45 // Formato: [ a1 a2 a3 ... an ]
46 void list_print(Node *list) {
47     printf("[ ");
48     while(list != NULL) {
49         printf("%d ", list->data);
50         list = list->next;
51     }
52     printf("]\n");
53 }
54
55 // Função que libera todos os nós
56 // alocados dinamicamente
57 void list_free(Node *list) {
58     while(list != NULL) {
59         Node *aux = list->next;
60         printf("list_free: nodo liberado: %d\n", list->data);
61         free(list);
62         list = aux;
63     }
64 }
65
66 // Função que imprime o tamanho da lista (número de nós)
67 size_t list_size(Node *list) {
68     size_t contador = 0;
69     for(Node *atual = list; atual != NULL; atual = atual->next) {
70         contador++;
71     }
72     return contador;
73 }
74
75 // Recebe como entrada um nó e devolve o valor dele.
76 int list_pegar_valor(Node *p) {
77     return p->data;
78 }
```

VPL

◀ [linked list] Implementar uma lista simplesmente encadeada – Parte 2

Seguir para...

[linked list] Implementar uma lista simplesmente encadeada – Parte 4 ▶

 Baixar o aplicativo móvel.