

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 4](#)[Descrição](#)[Enviar](#)[Editar](#)[Visualizar envios](#)

[linked list] Implementar uma lista simplesmente encadeada - Parte 4

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 recebe como entrada uma lista simplesmente encadeada e
// executa uma varredura na lista, comparando nós consecutivos dois-a-dois e movendo o maior
// elemento que ela encontra para a direita na lista.
// A função retorna como resultado a lista modificada.
// Por exemplo, se passarmos como entrada a lista [9,8,7,6,5] então a lista resultante deve ser
// [8,7,6,5,9].
// Veja que a função foi comparando os elementos dois-a-dois da esquerda para a direita, movendo o
// maior para a
// direita, nesse caso o número 9 que é o maior de todos na varredura acabou sendo movido para o
// final da lista.
// Mais um exemplo: se passarmos a lista [6,5,4,3,8,9] como entrada, então a lista resultante deve
// ser [5,4,3,6,8,9].
Node* list_varredura(Node *p);

// Esta função usa a função list_varredura descrita acima para implementar
// o algoritmo de ordenação por bolha (bubblesort) numa lista encadeada.
// Complexidade: O(n^2).
// Não é preciso alocar memória nessa função.
```

```
Node* list_bubblesort(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
show	mostra todas as listas na tela
bubblesort p	ordena a lista p usando o algoritmo de ordenação por bolhas
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio 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         // ----- varredura l -----
76         else if(strcmp(token, "varredura") == 0) {
77             int l = atoi(strtok(NULL, " "));
78             listas.data[l] = list_varredura(listas.data[l]);
79         }
80
81         // ----- bubblesort l -----
82         else if(strcmp(token, "bubblesort") == 0) {
83             int l = atoi(strtok(NULL, " "));
84             listas.data[l] = list_bubblesort(listas.data[l]);
85         }
86
87         // ----- addFront l a1 a2 ...
88         else if(strcmp(token, "addFront") == 0) {
89             int l = atoi(strtok(NULL, " "));
90             char *p = NULL;
91             while((p = strtok(NULL, " ")) != NULL) {
92                 int val = atoi(p);
93                 listas.data[l] = list_push_front(listas.data[l], val);
94             }
95         }
96
97         // ----- addBack l a1 a2 ...
98         else if(strcmp(token, "addBack") == 0) {
99             int l = atoi(strtok(NULL, " "));
100            char *p = NULL;
101            while((p = strtok(NULL, " ")) != NULL) {
102                int val = atoi(p);
103                listas.data[l] = list_push_back(listas.data[l], val);
104            }
105        }
106
107        // ----- show -----
108        else if(strcmp(token, "show") == 0) {
109            for(size_t i = 0; i < listas.size; i++) {
110                printf("lista %zu: ", i);
111                list_print(listas.data[i]);
112            }
113        }
114
115        else {
116            printf("comando inexistente\n");

```

```

117     }
118 }
119
120     return 0;
121 }
```

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 sentinel
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 // Esta função recebe como entrada uma lista simplesmente encadeada e
35 // executa uma varredura na lista, comparando nós consecutivos dois-a-dois e movendo o maior
36 // elemento que ela encontra para a direita na lista.
37 // A função retorna como resultado a lista modificada.
38 // Por exemplo, se passarmos como entrada a lista [9,8,7,6,5] então a lista resultante deve ser [8,7,6,5,9].
39 // Veja que a função foi comparando os elementos dois-a-dois da esquerda para a direita, movendo o maior para a
40 // direita, nesse caso o número 9 que é o maior de todos na varredura acabou sendo movido para o final da lista.
41 // Mais um exemplo: se passarmos a lista [6,5,4,3,8,9] como entrada, então a lista resultante deve ser [5,4,3,6,8,9].
42 Node* list_varredura(Node *p);
43
44 // Esta função usa a função list_varredura descrita acima para implementar
45 // o algoritmo de ordenação por bolha (bubblesort) numa lista encadeada.
46 // Complexidade: O(n^2).
47 // Não é preciso alocar memória nessa função.
48 Node* list_bubblesort(Node *p);
49
50 #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 }

```

VPL

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

Seguir para...

[Slides da aula: Variações de Listas Encadeadas ►](#)

©2020 – Universidade Federal do Ceará – Campus Quixadá.

Todos os direitos reservados.

Av. José de Freitas Queiroz, 5003

Cedro – Quixadá – Ceará CEP: 63902-580

Secretaria do Campus: (88) 3411-9422

[Baixar o aplicativo móvel.](#)