

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 2](#)

- ☰ Descrição
- 📁 Enviar
- 📄 Editar
- 📁 Visualizar envios

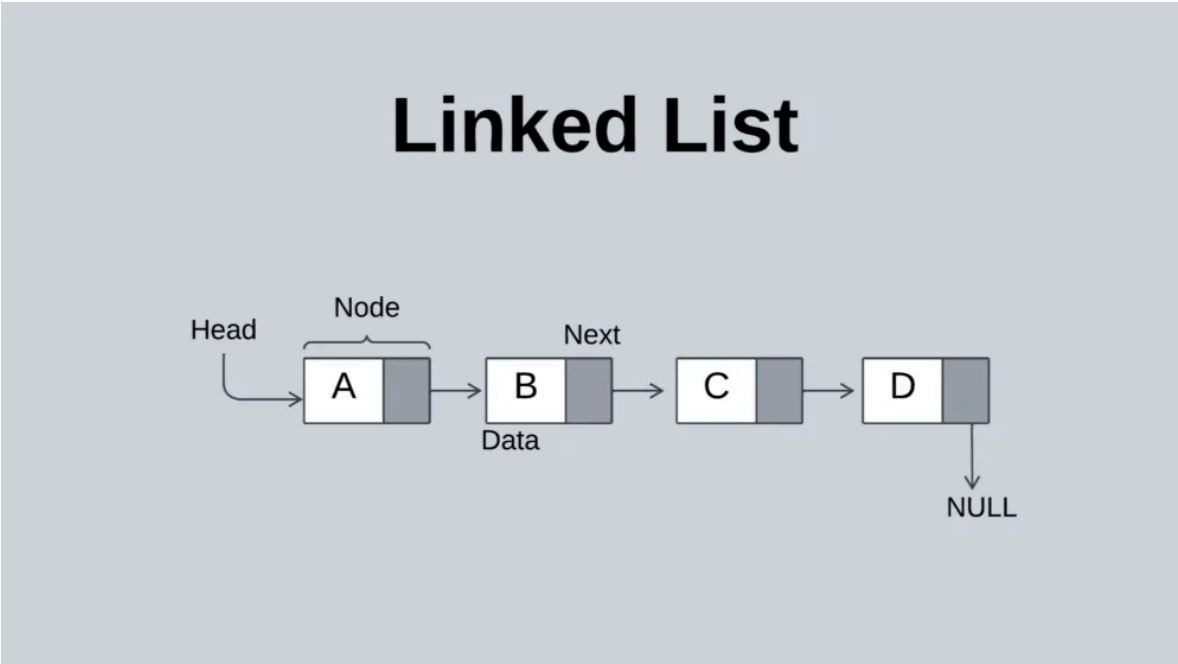
[linked list] Implementar uma lista simplesmente encadeada – Parte 2

📅 **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 reverte a ordem dos elementos na lista.
// Por exemplo, se você tiver a lista [10,30,40,50], então após
// chamar essa função a sua lista torna-se [50,40,30,10].
// A complexidade dessa função deve ser O(n). Além disso, faça essa
// função SEM ALOCAR NOVOS NODES. É possível fazer essa função sem
// alocar novos nós. Você precisará apenas de três ponteiros adicionais
// para te ajudar a reverter a sua lista.
// Retorna um ponteiro para a lista revertida.
Node* list_reverter(Node *p);

// Essa função recebe como parâmetro uma lista, um índice e um valor inteiro
// e insere esse inteiro na posição indicada pelo índice.
// Note que os índices estarão no intervalo [0..n-1].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_inserir_apos
// com o índice 2 e com o valor 999, essa função vai
// inserir o 999 logo após o elemento 3, resultando na lista: [1,2,3,999,4,5,6].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_inserir_apos(Node *p, size_t index, int val);
```

```
// Remove o elemento que está após o índice indicado.
// Note que os índices estarão no intervalo [0..n-2].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_remove_apos
// com o índice 4, essa função vai remover o número 6, resultando na lista: [1,2,3,4,5].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_remove_apos(Node *p, size_t index);
```

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
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio da lista k
show	mostra todas as listas na tela
addBack k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no final da lista k
insert_after k i v	insere o valor v na lista k após o índice i
remove_after k i	remove o valor após o índice i na lista k
reverse k	Inverte os elementos 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         // ----- insert_after l i v -----
76         else if(strcmp(token, "insert_after") == 0) {
77             int l = atoi(strtok(NULL, " "));
78             int i = atoi(strtok(NULL, " "));
79             int v = atoi(strtok(NULL, " "));
80             listas.data[l] = list_inserir_apos(listas.data[l], i, v);
81         }
82
83         // ----- remove_after l i -----
84         else if(strcmp(token, "remove_after") == 0) {
85             int l = atoi(strtok(NULL, " "));
86             int i = atoi(strtok(NULL, " "));
87             listas.data[l] = list_remover_apos(listas.data[l], i);
88         }
89
90         // ----- reverse l -----
91         else if(strcmp(token, "reverse") == 0) {
92             int l = atoi(strtok(NULL, " "));
93             listas.data[l] = list_reverter(listas.data[l]);
94         }
95
96         // ----- addFront l a1 a2 ... -----
97         else if(strcmp(token, "addFront") == 0) {
98             int l = atoi(strtok(NULL, " "));
99             char *p = NULL;
100             while((p = strtok(NULL, " ")) != NULL) {
101                 int val = atoi(p);
102                 listas.data[l] = list_push_front(listas.data[l], val);
103             }
104         }
105
106         // ----- addBack l a1 a2 ... -----
107         else if(strcmp(token, "addBack") == 0) {
108             int l = atoi(strtok(NULL, " "));
109             char *p = NULL;
110             while((p = strtok(NULL, " ")) != NULL) {
111                 int val = atoi(p);
112                 listas.data[l] = list_push_back(listas.data[l], val);
113             }
114         }
115
116         // ----- show -----
```

```
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 }
131
```

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 // Esta função reverte a ordem dos elementos na lista.
35 // Por exemplo, se você tiver a lista [10,30,40,50], então após
36 // chamar essa função a sua lista torna-se [50,40,30,10].
37 // A complexidade dessa função deve ser O(n). Além disso, faça essa
38 // função SEM ALOCAR NOVOS NODES. É possível fazer essa função sem
39 // alocar novos nós. Você precisará apenas de três ponteiros adicionais
40 // para te ajudar a reverter a sua lista.
41 // Retorna um ponteiro para a lista revertida.
42 Node* list_reverter(Node *p);
43
44 // Essa função recebe como parâmetro uma lista, um índice e um valor inteiro
45 // e insere esse inteiro na posição indicada pelo índice.
46 // Note que os índices estarão no intervalo [0..n-1].
47 // Ao final, retorna a lista modificada.
48 // Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_inserir_apos
49 // com o índice 2 e com o valor 999, essa função vai
50 // inserir o 999 logo após o elemento 3, resultando na lista: [1,2,3,999,4,5,6].
51 // Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
52 // maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
53 Node* list_inserir_apos(Node *p, size_t index, int val);
54
55 // Remove o elemento que está após o índice indicado.
56 // Note que os índices estarão no intervalo [0..n-2].
57 // Ao final, retorna a lista modificada.
58 // Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_remove_apos
59 // com o índice 4, essa função vai remover o número 6, resultando na lista: [1,2,3,4,5].
60 // Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
61 // maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
62 Node* list_remove_apos(Node *p, size_t index);
63
64
65 #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 1

Seguir para...

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