










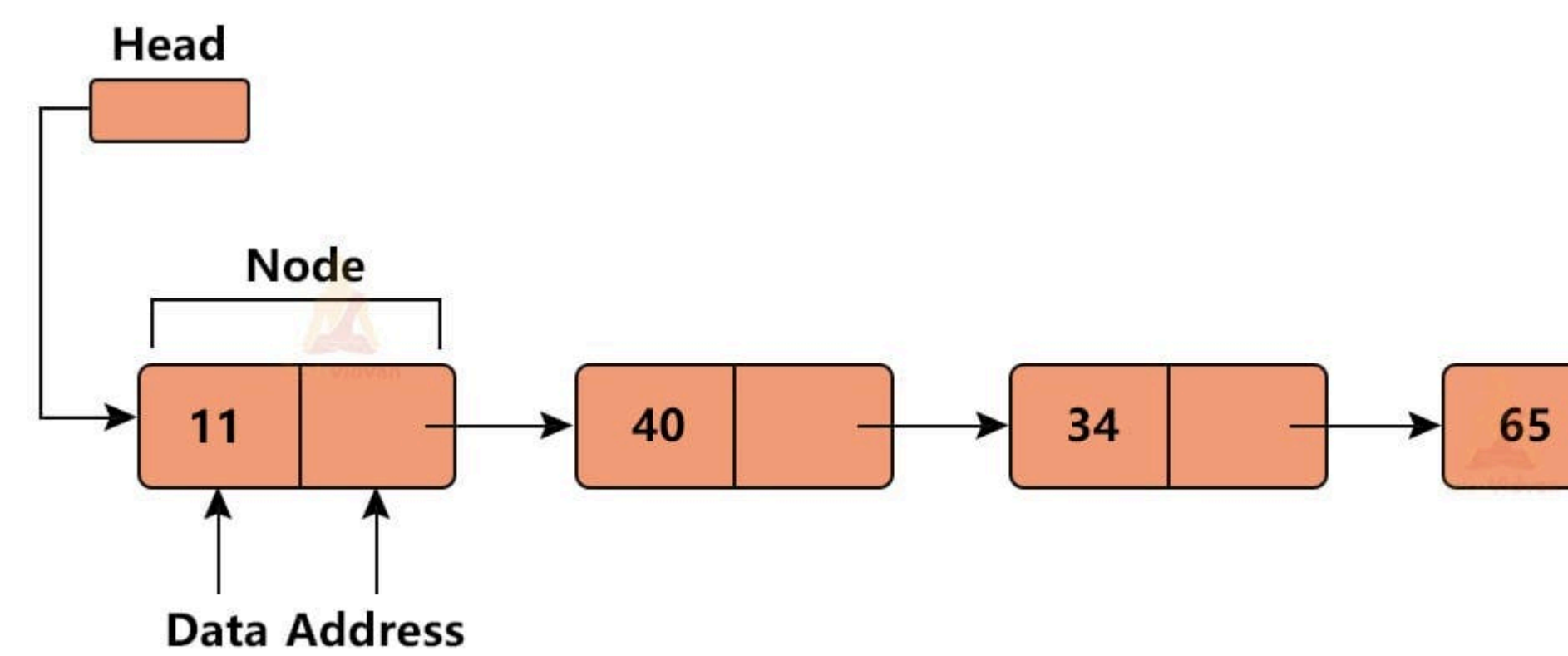
QXD0010 – ESTRUTURA DE DADOS – 01A – 2025.2

[Página inicial](#) ▶ [Meus cursos](#) ▶ [QXD0010 – ESTRUTURA DE DADOS – 01A – 2025.2](#) ▶ [Tópico 9. Lista Simplesmente Encadeada](#) ▶ [\[linked list\] Implementar uma lista simplesmente encadeada - Parte 1](#)

 Descrição  Enviar  Editar  Visualizar envios

[linked list] Implementar uma lista simplesmente encadeada – Parte 1

 **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 concatena duas listas simplesmente encadeadas passadas como parâmetro e
// retorna a lista resultante já concatenada.
// Exemplo se p1 = [1,2,3,4] e p2 = [44,66,77], então a lista resultante deve ser
// [1,2,3,4,44,66,77].
// Se a lista p1 for vazia, deve retornar p2.
Node* list_concatenar(Node *p1, Node *p2);
// Esta função recebe como entrada uma lista e faz uma
// cópia profunda (deep copy) da lista, i.e., a lista
// resultante contém os mesmos elementos que a original só que
// elas não compartilham nós. As duas listas são independentes.
// Ou seja, essa função cria um clone da lista original e retorna o clone.
Node* list_clone(Node *p);
// Esta função remove todas as ocorrências do valor 'val' da sua lista.
// Por exemplo, se você tiver a lista [4,1,2,4,4,3,4,4] e chamar
// essa função com o valor 4, a lista resultante sera [1,2,3].
// Não esqueça de liberar os nós que forem removidos.
```

```
// Antes de liberar o nó você deve imprimir "liberado: %d" e coloca o valor do nó liberado.
// Retorna a lista modificada.
Node* list_remove_todos(Node *p, int val);
// Esta função retorna true se as duas listas são idênticas; ou
// retorna false caso contrário.
// Duas listas são idênticas se elas têm os mesmos elementos na mesma ordem.
bool list_sao_iguais(Node* p1, Node *p2);
// Esta função retorna true se as duas listas são diferentes; ou
// retorna false caso contrário.
// Duas listas são diferentes se elas não são idênticas.
bool list_sao_diferentes(Node* p1, Node *p2);
```

Exemplo de Entrada

```
create
create
show
addFront 1 7 2 2 2 5 2 2 2
show
addFront 1 3 4 4 4
show
addFront 0 8 3 3 3 3 3 3 3 3
show
exit
```

Exemplo de Saída

```
$create
$create
$show
lista 0: [ ]
lista 1: [ ]
$addFront 1 7 2 2 2 5 2 2 2
$show
lista 0: [ ]
lista 1: [ 2 2 2 5 2 2 2 7 ]
$addFront 1 3 4 4 4
$show
lista 0: [ ]
lista 1: [ 4 4 4 3 2 2 2 5 2 2 2 7 ]
$addFront 0 8 3 3 3 3 3 3 3 3
$show
lista 0: [ 3 3 3 3 3 3 3 3 8 ]
lista 1: [ 4 4 4 3 2 2 2 5 2 2 2 7 ]
$exit
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 8
list_free: nodo liberado: 4
list_free: nodo liberado: 4
list_free: nodo liberado: 4
list_free: nodo liberado: 3
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 5
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 7
```

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
clone k	cria uma nova lista com os mesmos elementos da lista k já existente
show	mostra todas as listas na tela
different p q	retorna true se e somente se as listas p e q forem diferentes
remove x k	remove todos os valores x da lista k
equals p q	retorna true se e somente se as listas p e q forem iguais
reverse k	Inverte os elementos da lista k
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio da lista k
concat k p	concatena a lista p ao final 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         // ----- clone l -----
76         else if(strcmp(token, "clone") == 0) {
77             int l = atoi(strtok(NULL, " "));
78             vector_push_back(&listas, list_clone(listas.data[l]));
79         }
80
81         // ----- concat l1 l2 -----
82         else if(strcmp(token, "concat") == 0) {
83             int l1 = atoi(strtok(NULL, " "));
84             int l2 = atoi(strtok(NULL, " "));
85             listas.data[l1] = list_concatenar(listas.data[l1], listas.data[l2]);
86             listas.data[l2] = NULL;
87         }
88
89         // ----- remove x l -----
90         else if(strcmp(token, "remove") == 0) {
91             int x = atoi(strtok(NULL, " "));
92             int l = atoi(strtok(NULL, " "));
93             listas.data[l] = list_remove_todos(listas.data[l], x);
94         }
95
96         // ----- equals l1 l2 -----
97         else if(strcmp(token, "equals") == 0) {
98             int l1 = atoi(strtok(NULL, " "));
99             int l2 = atoi(strtok(NULL, " "));
100             if(list_sao_iguais(listas.data[l1], listas.data[l2]))
101                 printf("listas iguais\n");
102             else
103                 printf("listas diferentes\n");
104         }
105
106         // ----- different l1 l2 -----
107         else if(strcmp(token, "different") == 0) {
108             int l1 = atoi(strtok(NULL, " "));
109             int l2 = atoi(strtok(NULL, " "));
110             if(list_sao_diferentes(listas.data[l1], listas.data[l2]))
111                 printf("listas diferentes\n");
112             else
113                 printf("listas iguais\n");
114         }
115
116         // ----- reverse l -----
```

```
117         else if(strcmp(token, "reverse") == 0) {
118             int l = atoi(strtok(NULL, " "));
119             listas.data[l] = list_reverter(listas.data[l]);
120         }
121
122         // ----- addFront l a1 a2 ... -----
123         else if(strcmp(token, "addFront") == 0) {
124             int l = atoi(strtok(NULL, " "));
125             char *p = NULL;
126             while((p = strtok(NULL, " ")) != NULL) {
127                 int val = atoi(p);
128                 listas.data[l] = list_push_front(listas.data[l], val);
129             }
130         }
131
132         // ----- addBack l a1 a2 ... -----
133         else if(strcmp(token, "addBack") == 0) {
134             int l = atoi(strtok(NULL, " "));
135             char *p = NULL;
136             while((p = strtok(NULL, " ")) != NULL) {
137                 int val = atoi(p);
138                 listas.data[l] = list_push_back(listas.data[l], val);
139             }
140         }
141
142         // ----- show -----
143         else if(strcmp(token, "show") == 0) {
144             for(size_t i = 0; i < listas.size; i++) {
145                 printf("lista %zu: ", i);
146                 list_print(listas.data[i]);
147                 printf("\n");
148             }
149         }
150
151         else {
152             printf("comando inexistente\n");
153         }
154     }
155
156     return 0;
157 }
```

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 concatena duas listas simplesmente encadeadas passadas como parâmetro e
35 // retorna a lista resultante já concatenada.
36 // Exemplo se p1 = [1,2,3,4] e p2 = [44,66,77], então a lista resultante deve ser [1,2,3,4,44,66,77].
37 // Se a lista p1 for vazia, deve retornar p2.
38 Node* list_concatenar(Node *p1, Node *p2);
39
40 // Esta função recebe como entrada uma lista e faz uma
41 // cópia profunda (deep copy) da lista, i.e., a lista
42 // resultante contém os mesmos elementos que a original só que
43 // elas não compartilham nós. As duas listas são independentes.
44 // Ou seja, essa função cria um clone da lista original e retorna o clone.
45 Node* list_clone(Node *p);
46
47
48 // Esta função remove todas as ocorrencias do valor 'val' da sua lista.
49 // Por exemplo, se você tiver a lista [4,1,2,4,4,3,4,4] e chamar
50 // essa função com o valor 4, a lista resultante sera [1,2,3].
51 // Não esqueça de liberar os nós que forem removidos.
52 // Antes de liberar o nó você deve imprimir "liberado: %d" e coloca o valor do nó liberado.
53 // Retorna a lista modificada.
54 Node* list_remove_todos(Node *p, int val);
55
56
57 // Esta função retorna true se as duas listas são idênticas; ou
58 // retorna false caso contrário.
59 // Duas listas são idênticas se elas têm os mesmos elementos na mesma ordem.
60 bool list_sao_iguais(Node* p1, Node *p2);
61
62
63 // Esta função retorna true se as duas listas são diferentes; ou
64 // retorna false caso contrário.
65 // Duas listas são diferentes se elas não são idênticas.
66 bool list_sao_diferentes(Node* p1, Node *p2);
67
68 #endif
```

LinkedList.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include "LinkedList.h"
5
6  struct node {
7      int data;
8      struct node *next;
9  };
10
11
12  Node *list_create(void) {
13      return NULL;
14  }
15
16  Node *list_push_front(Node *list, int value) {
17      Node* novo = (Node*) malloc(sizeof(Node));
18      novo->data = value;
19      novo->next = list;
20      return novo;
21  }
22
23  Node *list_push_back(Node *list, int value) {
24      Node *novo = (Node *)malloc(sizeof(Node));
25      novo->data = value;
26      novo->next = NULL;
27
28      if(list == NULL) {
29          return novo;
30      } else {
31          Node *q = list;
32          while(q->next != NULL) {
33              q = q->next;
34          }
35          q->next = novo;
36          return list;
37      }
38  }
39
40  void list_print(Node *list) {
41      printf("[ ");
42      while(list != NULL) {
43          printf("%d ", list->data);
44          list = list->next;
45      }
46      printf("]\n");
47  }
48
49  void list_free(Node *list) {
50      while(list != NULL) {
51          Node *aux = list->next;
52          printf("list_free: nodo liberado: %d\n", list->data);
53          free(list);
54          list = aux;
55      }
56  }
57
58  size_t list_size(Node *list) {
59      size_t contador = 0;
60      for(Node *atual = list; atual != NULL; atual = atual->next) {
61          contador++;
62      }
63      return contador;
64  }
```

VPL

◀ Código desenvolvido na aula do dia 25/11/2025

Seguir para...

[linked list] Implementar uma lista simplesmente encadeada - Parte 2 ▶

©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.