

Felipe Barroso de Castro  
RA: 2311292  
Curso: Engenharia de Software

## Lista Encadeada Simples

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Estruturas definidas
5  typedef struct no {
6      int dado;
7      struct no *prox;
8  } No;
9
10 typedef struct lista {
11     No *inicio;
12     No *fim;
13     unsigned int length;
14 } Lista;
```

```
15
16 // Função para inserir no início
17 void inserirInicio(Lista *lista, int valor) {
18     No *novo_no = (No *)malloc(sizeof(No));
19     if (novo_no == NULL) {
20         printf("Erro ao alocar memória\n");
21         return;
22     }
23     novo_no->dado = valor;
24     novo_no->prox = lista->inicio;
25     lista->inicio = novo_no;
26
27     if (lista->fim == NULL) {
28         lista->fim = novo_no;
29     }
30     lista->length++;
31 }
```

```
33 // Função para inserir no fim
34 ✓ void inserirFim(Lista *lista, int valor) {
35     No *novo_no = (No *)malloc(sizeof(No));
36 ✓     if (novo_no == NULL) {
37         printf("Erro ao alocar memória\n");
38         return;
39     }
40     novo_no->dado = valor;
41     novo_no->prox = NULL;
42
43 ✓     if (lista->fim != NULL) {
44         lista->fim->prox = novo_no;
45 ✓     } else {
46         lista->inicio = novo_no;
47     }
48
49     lista->fim = novo_no;
50     lista->length++;
51 }
```

```
// Função para inserir de forma ordenada
void inserirOrdenado(Lista *lista, int valor) {
    No *novo_no = (No *)malloc(sizeof(No));
    if (novo_no == NULL) {
        printf("Erro ao alocar memória\n");
        return;
    }
    novo_no->dado = valor;

    if (lista->inicio == NULL || lista->inicio->dado >= valor) {
        novo_no->prox = lista->inicio;
        lista->inicio = novo_no;
        if (lista->fim == NULL) {
            lista->fim = novo_no;
        }
    } else {
        No *atual = lista->inicio;
        while (atual->prox != NULL && atual->prox->dado < valor) {
            atual = atual->prox;
        }
        novo_no->prox = atual->prox;
        atual->prox = novo_no;
        if (novo_no->prox == NULL) {
            lista->fim = novo_no;
        }
    }
    lista->length++;
}
```

```

// Função para deletar do início
void deletarInicio(Lista *lista) {
    if (lista->inicio == NULL) {
        printf("Lista está vazia\n");
        return;
    }

    No *temp = lista->inicio;
    lista->inicio = lista->inicio->prox;
    free(temp);

    if (lista->inicio == NULL) {
        lista->fim = NULL;
    }
    lista->length--;
}

```

## Lista Duplamente Encadeada

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Estruturas definidas
5  typedef struct no {
6      char dado;
7      struct no *ant;
8      struct no *prox;
9  } No;
10
11 typedef struct listaDEnc {
12     No *inicio;
13     No *fim;
14     unsigned int length;
15 } ListaDEnc;

```

```

17 // Função para deletar do início
18 void deletarInicio(ListaDEnc *lista) {
19     if (lista->inicio == NULL) {
20         printf("Lista está vazia\n");
21         return;
22     }
23
24     No *temp = lista->inicio;
25     lista->inicio = lista->inicio->prox;
26
27     if (lista->inicio != NULL) {
28         lista->inicio->ant = NULL;
29     } else {
30         lista->fim = NULL;
31     }
32
33     free(temp);
34     lista->length--;
35 }
36
37 // Função para deletar do fim
38 void deletarFim(ListaDEnc *lista) {
39     if (lista->fim == NULL) {
40         printf("Lista está vazia\n");
41         return;
42     }
43
44     No *temp = lista->fim;
45     lista->fim = lista->fim->ant;
46
47     if (lista->fim != NULL) {
48         lista->fim->prox = NULL;
49     } else {
50         lista->inicio = NULL;
51     }
52
53     free(temp);
54     lista->length--;
55 }
56

```

## Funções de Pilha

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Estruturas definidas
5  typedef struct no {
6      unsigned int dado;
7      struct no *prox;
8  } No;
9
10 typedef struct pilha {
11     No *topo;
12     unsigned int length;
13 } Pilha;
```

```
15 // Função para retornar o valor do topo
16 ✓ int topo(Pilha *pilha) {
17     if (pilha->topo == NULL) {
18         return -1;
19     }
20     return pilha->topo->dado;
21 }
```

```
23 // Função para empilhar
24 ✓ void empilha(Pilha *pilha, unsigned int dado) {
25     No *novo_no = (No *)malloc(sizeof(No));
26     if (novo_no == NULL) {
27         printf("Erro ao alocar memória\n");
28         return;
29     }
30     novo_no->dado = dado;
31     novo_no->prox = pilha->topo;
32     pilha->topo = novo_no;
33     pilha->length++;
34 }
```

```
36 // Função para desempilhar
37 ✓ int pop(Pilha *pilha) {
38 ✓     if (pilha->topo == NULL) {
39         return -1;
40     }
```

```
41
42     No *temp = pilha->topo;
43     int dado = temp->dado;
44     pilha->topo = pilha->topo->prox;
45     free(temp);
46     pilha->length--;
47
48     return dado;
49 }
50
```

## Funções de Fila



```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Estruturas definidas
5  typedef struct no {
6      int dado;
7      struct no *prox;
8  } No;
9
10 typedef struct fila {
11     No *inicio;
12     No *fim;
13     unsigned int length;
14 } Fila;
```

```
16 // Função para enfileirar
17 void enfileira(Fila *fila, int valor) {
18     No *novo_no = (No *)malloc(sizeof(No));
19     if (novo_no == NULL) {
20         printf("Erro ao alocar memória\n");
21         return;
22     }
23     novo_no->dado = valor;
24     novo_no->prox = NULL;
25
26     if (fila->fim != NULL) {
27         fila->fim->prox = novo_no;
28     } else {
29         fila->inicio = novo_no;
30     }
31
32     fila->fim = novo_no;
33     fila->length++;
34 }
```

```
35
36 // Função para desenfileirar
37 int desenfileira(Fila *fila) {
38     if (fila->inicio == NULL) {
39         return -1;
40     }
41
42     No *temp = fila->inicio;
43     int valor = temp->dado;
44     fila->inicio = fila->inicio->prox;
45
46     if (fila->inicio == NULL) {
47         fila->fim = NULL;
48     }
49
50     free(temp);
51     fila->length--;
52
53     return valor;
54 }
```