

## roteiro-05/ex01-01.h

```
1  #ifndef FILA_SEQUENCIAL_H
2  #define FILA_SEQUENCIAL_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 100
8
9  typedef struct {
10     int qtd, ini, fim;
11     int dados[MAX];
12 } Fila;
13
14 Fila* criaFila() {
15     Fila* fi;
16     fi = (Fila*)malloc(sizeof(Fila));
17     if (fi != NULL) {
18         fi->qtd = fi->ini = fi->fim = 0;
19     }
20     return fi;
21 }
22
23 void destroiFila(Fila** fi) {
24     if (*fi != NULL)
25         free(*fi);
26     *fi = NULL;
27 }
28
29 int tamanhoFila(Fila* fi) {
30     if (fi == NULL)
31         return -1;
32     return fi->qtd;
33 }
34
35 int estaCheia(Fila* fi) {
36     if (fi == NULL)
37         return -1;
38     return (fi->qtd == MAX);
39 }
40
41 int estaVazia(Fila* fi) {
42     if (fi == NULL)
43         return -1;
44     return (fi->qtd == 0);
45 }
46
47 int enfileirar(Fila* fi, int elem) {
48     if (fi == NULL) return 0;
49     if (estaCheia(fi)) return 0;
50     fi->dados[fi->fim] = elem;
51     fi->fim = (fi->fim + 1) % MAX;
52     fi->qtd++;
53     return 1;
54 }
55
```

```
56 int desenfileirar(Fila* fi) {
57     if (fi == NULL) return 0;
58     if (estaVazia(fi)) return 0;
59     fi->ini = (fi->ini + 1) % MAX;
60     fi->qtd--;
61     return 1;
62 }
63
64 int verInicio(Fila* fi, int* p) {
65     if (fi == NULL) return 0;
66     if (estaVazia(fi)) return 0;
67     *p = fi->dados[fi->ini];
68     return 1;
69 }
70
71 void imprime(Fila* fi) {
72     if (fi == NULL) return;
73     if (estaVazia(fi)) {
74         printf("Fila Vazia!\n");
75         return;
76     }
77     int i = fi->ini;
78     printf("Elementos: \n");
79     do {
80         printf("%d ", fi->dados[i]);
81         i = (i + 1) % MAX;
82     } while (i != fi->fim);
83     printf("\n");
84 }
85
86 #endif
```

## roteiro-05/ex01-02.h

```
1  #ifndef FILA_ENCADEADA_H
2  #define FILA_ENCADEADA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO {
8      int info;
9      struct NO* prox;
10 } NO;
11
12 typedef struct {
13     int qtd;
14     struct NO *ini, *fim;
15 } Fila;
16
17 Fila* criaFila() {
18     Fila* nova = (Fila*)malloc(sizeof(Fila));
19     nova->qtd = 0;
20     nova->ini = NULL;
21     nova->fim = NULL;
22
23     return nova;
24 }
25
26 void destroiFila(Fila** fi) {
27     if (*fi != NULL) {
28         free(*fi);
29         *fi = NULL;
30     }
31 }
32
33 NO* criaNo() {
34     NO* novo = (NO*)malloc(sizeof(NO));
35     novo->info = 0;
36     novo->prox = NULL;
37
38     return novo;
39 }
40
41 void destroiNo(NO* no) {
42     free(no);
43 }
44
45 int tamanhoFila(Fila* fi) {
46     if (fi == NULL) return 0;
47     return fi->qtd;
48 }
49
50 int estaVazia(Fila* fi) {
51     return (fi == NULL || fi->ini == NULL);
52 }
53
```

```

54 int enfileirar(Fila* fi, int elem) {
55     if (fi == NULL) return 0;
56
57     NO* novo = criaNo();
58     novo->info = elem;
59
60     if (estaVazia(fi)) {
61         fi->ini = novo;
62         fi->fim = novo;
63     } else {
64         fi->fim->prox = novo;
65         fi->fim = novo;
66     }
67     fi->qtd++;
68
69     return 1;
70 }
71
72 int desenfileirar(Fila* fi) {
73     if (fi == NULL || estaVazia(fi)) return 0;
74
75     NO* aux = fi->ini;
76     fi->ini = fi->ini->prox;
77     if (estaVazia(fi)) fi->fim = NULL;
78
79     destroiNo(aux);
80     fi->qtd--;
81
82     return 1;
83 }
84
85 int verInicio(Fila* fi, int* p) {
86     if (fi == NULL || estaVazia(fi)) return 0;
87
88     *p = fi->ini->info;
89
90     return 1;
91 }
92
93 void imprime(Fila* fi) {
94     if (fi == NULL) return;
95     if (estaVazia(fi)) {
96         printf("Fila vazia");
97         return;
98     }
99
100     printf("Elementos:\n");
101     NO* i = fi->ini;
102     do {
103         printf("%d ", i->info);
104         i = i->prox;
105     } while (i != NULL);
106     printf("\n");
107 }
108
109 #endif

```

## roteiro-05/ex01.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*
5   *
6   *  O ARQUIVO FUNCIONA PARA AS DUAS IMPLEMENTACOES DE FILA
7   *  BASTA IMPORTAR APENAS A BIBLIOTECA DESEJADA
8   *
9   *  EX01-01 = LISTA SEQUENCIAL ESTATICA
10  *  EX01-02 = LISTA SIMPLEMENTE ENCADEADA
11  *
12  */
13
14  // #include "ex01-01.h"
15  #include "ex01-02.h"
16
17  enum {
18      EXIT = 0,
19      CREATE,
20      QUEUE,
21      START,
22      DEQUEUE,
23      PRINT,
24      DESTROY
25  } Options;
26
27  int getOption() {
28      int option;
29
30      printf("\n=====\\n");
31      printf("(%) Criar\\n", CREATE);
32      printf("(%) Enfileirar\\n", QUEUE);
33      printf("(%) Ver inicio\\n", START);
34      printf("(%) Desenfileirar\\n", DEQUEUE);
35      printf("(%) Imprimir\\n", PRINT);
36      printf("(%) Destruir\\n", DESTROY);
37      printf("(%) Sair\\n", EXIT);
38      printf("=====\\n");
39      printf("Operacao: ");
40
41      scanf("%d", &option);
42      printf("\\n");
43
44      return option;
45  }
46
47  int runMenu() {
48      Fila* queue = NULL;
49      int exit = 0, item;
50
51      do {
52          switch (getOption()) {
53              case CREATE:
54                  if (queue != NULL) {
55                      destroiFila(&queue);
56                      printf("Fila resetada");
57                  }
```

```

58         queue = criaFila();
59         break;
60
61     case QUEUE:
62         printf("Elemento para enfileirar: ");
63         scanf("%d", &item);
64         if (enfileirar(queue, item)) {
65             printf("Enfileirou (%d)", item);
66         } else {
67             printf("Nao foi possivel enfileirar (%d)", item);
68         }
69         break;
70
71     case START:
72         if (verInicio(queue, &item)) {
73             printf("Inicio da fila = %d\n", item);
74         } else {
75             printf("Nao foi possivel ver o inicio da fila");
76         }
77         break;
78
79     case DEQUEUE:
80         if (desenfileirar(queue)) {
81             printf("Desenfileirou o primeiro elemento");
82         } else {
83             printf("Nao foi possivel desenfileirar");
84         }
85         break;
86
87     case PRINT:
88         imprime(queue);
89         break;
90
91     case DESTROY:
92         destroiFila(&queue);
93         printf("Fila destruida");
94         break;
95
96     case EXIT:
97         if (queue != NULL) {
98             destroiFila(&queue);
99         }
100         printf("Programa encerrado");
101         exit = 1;
102         break;
103
104     default:
105         printf("Opcao desconhecida, tente novamente");
106     }
107     printf("\n");
108 } while (!exit);
109 }
110
111 int main() {
112     runMenu();
113     return 0;
114 }

```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-05
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-05$ ./ex01

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 1

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para enfileirar: 1
Enfileirou (1)

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para enfileirar: 2
Enfileirou (2)

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para enfileirar: 3
Enfileirou (3)

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 5

Elementos:
1 2 3

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 4

Desenfileirou o primeiro elemento

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 5

Elementos:
2 3

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 6

Fila destruida

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-05$
```

## roteiro-05/ex02-01.h

```
1  #ifndef PILHA_SEQUENCIAL_H
2  #define PILHA_SEQUENCIAL_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 100
8
9  typedef struct {
10     int topo;
11     int dados[MAX];
12 } Pilha;
13
14 Pilha* criaPilha() {
15     Pilha* pi;
16     pi = (Pilha*)malloc(sizeof(Pilha));
17     if (pi != NULL) {
18         pi->topo = 0;
19     }
20     return pi;
21 }
22
23 void destroiPilha(Pilha** pi) {
24     if (*pi != NULL) {
25         free(*pi);
26         *pi = NULL;
27     }
28 }
29
30 int tamanhoPilha(Pilha* pi) {
31     if (pi == NULL)
32         return -1;
33     return pi->topo;
34 }
35
36 int estaCheia(Pilha* pi) {
37     if (pi == NULL)
38         return -1;
39     return (pi->topo == MAX);
40 }
41
42 int estaVazia(Pilha* pi) {
43     if (pi == NULL)
44         return -1;
45     return (pi->topo == 0);
46 }
47
48 int empilhar(Pilha* pi, int elem) {
49     if (pi == NULL) return 0;
50     if (estaCheia(pi)) return 0;
51     pi->dados[pi->topo] = elem;
52     pi->topo++;
53     return 1;
54 }
55
```



```
56 int desempilhar(Pilha* pi) {
57     if (pi == NULL) return 0;
58     if (estaVazia(pi)) return 0;
59     pi->topo--;
60     return 1;
61 }
62
63 int verTopo(Pilha* pi, int* p) {
64     if (pi == NULL) return 0;
65     if (estaVazia(pi)) return 0;
66     *p = pi->dados[pi->topo - 1];
67     return 1;
68 }
69
70 void imprime(Pilha* pi) {
71     if (pi == NULL) return;
72     if (estaVazia(pi)) {
73         printf("Pilha Vazia!\n");
74         return;
75     }
76     printf("Elementos: \n");
77     int i;
78     for (i = pi->topo - 1; i >= 0; i--)
79         printf("%d ", pi->dados[i]);
80     printf("\n");
81 }
82
83 #endif
```

## roteiro-05/ex02-02.h

```
1  #ifndef PILHA_ENCADEADA_H
2  #define PILHA_ENCADEADA_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO {
8      int info;
9      struct NO* prox;
10 } NO;
11
12 typedef struct {
13     int qtd;
14     struct NO* topo;
15 } Pilha;
16
17 Pilha* criaPilha() {
18     Pilha* pi = (Pilha*)malloc(sizeof(Pilha));
19     pi->topo = NULL;
20     pi->qtd = 0;
21
22     return pi;
23 }
24
25 void destroiPilha(Pilha** pi) {
26     if (*pi != NULL) {
27         free(*pi);
28         *pi = NULL;
29     }
30 }
31
32 NO* criaNo() {
33     NO* novo = (NO*)malloc(sizeof(NO));
34     novo->info = 0;
35     novo->prox = NULL;
36
37     return novo;
38 }
39
40 void destroiNo(NO* no) {
41     free(no);
42 }
43
44 int tamanhoPilha(Pilha* pi) {
45     if (pi == NULL)
46         return 0;
47     return pi->qtd;
48 }
49
50 int estaVazia(Pilha* pi) {
51     return (pi == NULL || pi->topo == NULL);
52 }
53
```

```

54 int empilhar(Pilha* pi, int elem) {
55     if (pi == NULL) return 0;
56
57     NO* novo = criaNo();
58     novo->info = elem;
59     novo->prox = pi->topo;
60     pi->topo = novo;
61     pi->qtd++;
62
63     return 1;
64 }
65
66 int desempilhar(Pilha* pi) {
67     if (pi == NULL || estaVazia(pi)) return 0;
68
69     NO* aux = pi->topo;
70     pi->topo = pi->topo->prox;
71
72     destroiNo(aux);
73     pi->qtd--;
74
75     return 1;
76 }
77
78 int verTopo(Pilha* pi, int* p) {
79     if (pi == NULL || estaVazia(pi)) return 0;
80
81     *p = pi->topo->info;
82
83     return 1;
84 }
85
86 void imprime(Pilha* pi) {
87     if (pi == NULL) return;
88     if (estaVazia(pi)) {
89         printf("Pilha Vazia!\n");
90         return;
91     }
92
93     printf("Elementos: \n");
94     NO* i = pi->topo;
95     do {
96         printf("%d ", i->info);
97         i = i->prox;
98     } while (i != NULL);
99     printf("\n");
100 }
101
102 #endif

```

## roteiro-05/ex02.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*
5   *
6   *  O ARQUIVO FUNCIONA PARA AS DUAS IMPLEMENTACOES DE FILA
7   *  BASTA IMPORTAR APENAS A BIBLIOTECA DESEJADA
8   *
9   *  EX02-01 = PILHA SEQUENCIAL ESTATICA
10  *  EX02-02 = PILHA SIMPLEMENTE ENCADEADA
11  *
12  */
13
14  // #include "ex02-01.h"
15  #include "ex02-02.h"
16
17  enum {
18      EXIT = 0,
19      CREATE,
20      STACK,
21      START,
22      UNSTACK,
23      PRINT,
24      DESTROY
25  } Options;
26
27  int getOption() {
28      int option;
29
30      printf("\n=====\\n");
31      printf("(%) Criar\\n", CREATE);
32      printf("(%) Empilhar\\n", STACK);
33      printf("(%) Ver topo\\n", START);
34      printf("(%) Desempilhar\\n", UNSTACK);
35      printf("(%) Imprimir\\n", PRINT);
36      printf("(%) Destruir\\n", DESTROY);
37      printf("(%) Sair\\n", EXIT);
38      printf("=====\\n");
39      printf("Operacao: ");
40
41      scanf("%d", &option);
42      printf("\\n");
43
44      return option;
45  }
46
47  int runMenu() {
48      Pilha* stack = NULL;
49      int exit = 0, item;
50
51      do {
52          switch (getOption()) {
53              case CREATE:
54                  if (stack != NULL) {
55                      destroiPilha(&stack);
56                      printf("Pilha resetada");
57                  }
```

```

58         stack = criaPilha();
59         break;
60
61     case STACK:
62         printf("Elemento para empilhar: ");
63         scanf("%d", &item);
64
65         if (empilhar(stack, item)) {
66             printf("Empilhou (%d)", item);
67         } else {
68             printf("Nao foi possivel empilhar (%d)", item);
69         }
70         break;
71
72     case START:
73         if (verTopo(stack, &item)) {
74             printf("Topo da pilha = %d\n", item);
75         } else {
76             printf("Nao foi possivel ver o topo da pilha");
77         }
78         break;
79
80     case UNSTACK:
81         if (desempilhar(stack)) {
82             printf("Desempilhou o primeiro elemento");
83         } else {
84             printf("Nao foi possivel desempilhar");
85         }
86         break;
87
88     case PRINT:
89         imprime(stack);
90         break;
91
92     case DESTROY:
93         destroiPilha(&stack);
94         printf("Pilha destruida");
95         break;
96
97     case EXIT:
98         if (stack != NULL) {
99             destroiPilha(&stack);
100         }
101         printf("Programa encerrado");
102         exit = 1;
103         break;
104
105     default:
106         printf("Opcao desconhecida, tente novamente");
107     }
108     printf("\n");
109 } while (!exit);
110 }
111
112 int main() {
113     runMenu();
114     return 0;
115 }

```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-05
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-05$ ./ex02

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 1

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para empilhar: 1
Empilhou (1)

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para empilhar: 2
Empilhou (2)

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 2

Elemento para empilhar: 3
Empilhou (3)

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 5

Elementos:
3 2 1

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 4

Desempilhou o primeiro elemento

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 5

Elementos:
2 1

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 6

Pilha destruida

=====
(1) Criar
(2) Empilhar
(3) Ver topo
(4) Desempilhar
(5) Imprimir
(6) Destruir
(0) Sair
=====
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-05$
```