

roteiro-06/ex01-01.h

```
1  #ifndef DEQUE_H
2  #define DEQUE_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 } Deque;
13
14 Deque* criaDeque() {
15     Deque* dq;
16     dq = (Deque*)malloc(sizeof(Deque));
17     if (dq != NULL) {
18         dq->qtd = dq->ini = dq->fim = 0;
19     }
20     return dq;
21 }
22
23 void destroiDeque(Deque** dq) {
24     if (*dq != NULL) {
25         free(*dq);
26         *dq = NULL;
27     }
28 }
29
30 int tamanhoDeque(Deque* dq) {
31     if (dq == NULL)
32         return -1;
33     return dq->qtd;
34 }
35
36 int estaCheio(Deque* dq) {
37     if (dq == NULL)
38         return -1;
39     return (dq->qtd == MAX);
40 }
41
42 int estaVazio(Deque* dq) {
43     if (dq == NULL)
44         return -1;
45     return (dq->qtd == 0);
46 }
47
48 int insereInicio(Deque* dq, int elem) {
49     if (dq == NULL) return 0;
50     if (estaCheio(dq)) return 0;
51     dq->ini = (dq->ini - 1 < 0 ? MAX - 1 : dq->ini - 1);
52     dq->dados[dq->ini] = elem;
53     dq->qtd++;
54     return 1;
55 }
56
```

```

57 int insereFim(Deque* dq, int elem) {
58     if (dq == NULL) return 0;
59     if (estaCheio(dq)) return 0;
60     dq->dados[dq->fim] = elem;
61     dq->fim = (dq->fim + 1) % MAX;
62     dq->qtd++;
63     return 1;
64 }
65
66 int removeInicio(Deque* dq) {
67     if (dq == NULL) return 0;
68     if (estaVazio(dq)) return 0;
69     dq->ini = (dq->ini + 1) % MAX;
70     dq->qtd--;
71     return 1;
72 }
73
74 int removeFim(Deque* dq) {
75     if (dq == NULL) return 0;
76     if (estaVazio(dq)) return 0;
77     dq->fim = (dq->fim - 1 < 0 ? MAX - 1 : dq->fim - 1);
78     dq->qtd--;
79     return 1;
80 }
81
82 int verInicio(Deque* dq, int* p) {
83     if (dq == NULL) return 0;
84     if (estaVazio(dq)) return 0;
85     *p = dq->dados[dq->ini];
86     return 1;
87 }
88
89 int verFim(Deque* dq, int* p) {
90     if (dq == NULL) return 0;
91     if (estaVazio(dq)) return 0;
92     int i = (dq->fim - 1 < 0 ? MAX - 1 : dq->fim - 1);
93     *p = dq->dados[i];
94     return 1;
95 }
96
97 void imprime(Deque* dq) {
98     if (dq == NULL) return;
99     if (estaVazio(dq)) {
100         printf("Deque Vazio!\n");
101         return;
102     }
103     int i = dq->ini;
104     printf("Elementos: \n");
105     do {
106         printf("%d ", dq->dados[i]);
107         i = (i + 1) % MAX;
108     } while (i != dq->fim);
109     // Usar do..while garante a impressao de todos elementos
110     // mesmo com a Deque cheia
111     printf("\n");
112 }
113
114 #endif

```

roteiro-06/ex01-02.h

```
1  #ifndef DDE_H
2  #define DDE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO{
8      int info;
9      struct NO* prox;
10     struct NO* ant;
11 }NO;
12
13 typedef struct{
14     int qtd;
15     struct NO* ini;
16     struct NO* fim;
17 }Deque;
18
19
20 NO* alocarNO(){
21     return (NO*) malloc (sizeof(NO));
22 }
23
24 void liberarNO(NO* q){
25     free(q);
26 }
27
28 Deque* criaDeque(){
29     Deque* dq;
30     dq = (Deque*) malloc (sizeof(Deque));
31     if(dq != NULL){
32         dq->qtd = 0;
33         dq->ini = NULL;
34         dq->fim = NULL;
35     }
36     return dq;
37 }
38
39 void destroiDeque(Deque **dq){
40     if(*dq != NULL){
41         NO* aux;
42         while((*dq)->ini != NULL){
43             aux = (*dq)->ini;
44             (*dq)->ini = (*dq)->ini->prox;
45             liberarNO(aux);
46         }
47         free(*dq);
48         *dq = NULL;
49     }
50 }
51
52 int tamanhoDeque(Deque *dq){
53     if(dq == NULL)
54         return -1;
55     return dq->qtd;
56 }
```

```

57
58 int estaVazio(Deque *dq){
59     if(dq == NULL)
60         return -1;
61     return (dq->qtd == 0);
62 }
63
64 int insereInicio(Deque* dq, int elem){
65     if(dq == NULL) return 0;
66     NO* novo = alocarNO();
67     if(novo == NULL) return 0;
68     novo->info = elem;
69     novo->ant = NULL;
70     if(estaVazio(dq)){
71         novo->prox = NULL;
72         dq->fim = novo;
73     }else{
74         dq->ini->ant = novo;
75         novo->prox = dq->ini;
76     }
77     dq->ini = novo;
78     dq->qtd++;
79     return 1;
80 }
81
82 int insereFim(Deque* dq, int elem){
83     if(dq == NULL) return 0;
84     NO* novo = alocarNO();
85     if(novo == NULL) return 0;
86     novo->info = elem;
87     novo->prox = NULL;
88     if(estaVazio(dq)){
89         novo->ant = NULL;
90         dq->ini = novo;
91     }else{
92         dq->fim->prox = novo;
93         novo->ant = dq->fim;
94     }
95     dq->fim = novo;
96     dq->qtd++;
97     return 1;
98 }
99
100 int removeInicio(Deque* dq){
101     if(dq == NULL) return 0;
102     if(estaVazio(dq)) return 0;
103     NO* aux = dq->ini;
104     if(dq->ini == dq->fim){
105         dq->ini = dq->fim = NULL;
106     }else{
107         dq->ini = dq->ini->prox;
108         dq->ini->ant = NULL;
109     }
110     liberarNO(aux);
111     dq->qtd--;
112     return 1;
113 }
114
115 int removeFim(Deque* dq){

```

```
116     if(dq == NULL) return 0;
117     if(estaVazio(dq)) return 0;
118     NO* aux = dq->fim;
119     if(dq->ini == dq->fim){
120         dq->ini = dq->fim = NULL;
121     }else{
122         dq->fim = dq->fim->ant;
123         dq->ini->prox = NULL;
124     }
125     liberarNO(aux);
126     dq->qtd--;
127     return 1;
128 }
129
130 int verInicio(Deque* dq, int* p){
131     if(dq == NULL) return 0;
132     if(estaVazio(dq)) return 0;
133     *p = dq->ini->info;
134     return 1;
135 }
136
137 int verFim(Deque* dq, int* p){
138     if(dq == NULL) return 0;
139     if(estaVazio(dq)) return 0;
140     *p = dq->fim->info;
141     return 1;
142 }
143
144 void imprime(Deque* dq){
145     if(dq == NULL) return;
146     if(estaVazio(dq)){
147         printf("Deque Vazio!\n");
148         return;
149     }
150     NO* aux = dq->ini;
151     printf("Elementos:\n");
152     while(aux != NULL){
153         printf("%d ", aux->info);
154         aux = aux->prox;
155     }
156     printf("\n");
157 }
158
159 #endif
```

roteiro-06/ex01.c

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

```

57 do {
58     switch (getOption()) {
59         case CREATE:
60             if (deque != NULL) {
61                 destroiDeque(&deque);
62                 printf("Deque resetado");
63             }
64             deque = criaDeque();
65             break;
66
67         case QUEUE_START:
68             printf("Elemento para enfileirar no inicio: ");
69             scanf("%d", &item);
70             if (insereInicio(deque, item)) {
71                 printf("Enfileirou (%d) no inicio", item);
72             } else {
73                 printf("Nao foi possivel enfileirar (%d)", item);
74             }
75             break;
76
77         case QUEUE_END:
78             printf("Elemento para enfileirar no fim: ");
79             scanf("%d", &item);
80             if (insereFim(deque, item)) {
81                 printf("Enfileirou (%d) no fim", item);
82             } else {
83                 printf("Nao foi possivel enfileirar (%d)", item);
84             }
85             break;
86
87         case START:
88             if (verInicio(deque, &item)) {
89                 printf("Inicio do deque = %d\n", item);
90             } else {
91                 printf("Nao foi possivel ver o inicio do deque");
92             }
93             break;
94
95         case END:
96             if (verFim(deque, &item)) {
97                 printf("Fim do deque = %d\n", item);
98             } else {
99                 printf("Nao foi possivel ver o fim do deque");
100             }
101             break;
102
103         case DEQUEUE_START:
104             if (removeInicio(deque)) {
105                 printf("Desenfileirou o primeiro elemento");
106             } else {
107                 printf("Nao foi possivel desenfileirar");
108             }
109             break;
110
111         case DEQUEUE_END:
112             if (removeFim(deque)) {
113                 printf("Desenfileirou o ultimo elemento");
114             } else {
115                 printf("Nao foi possivel desenfileirar");

```

```
116         }
117         break;
118
119     case PRINT:
120         imprime(deque);
121         break;
122
123     case DESTROY:
124         destroiDeque(&deque);
125         printf("Deque destruido");
126         break;
127
128     case EXIT:
129         if (deque != NULL) {
130             destroiDeque(&deque);
131         }
132         printf("Programa encerrado");
133         exit = 1;
134         break;
135
136     default:
137         printf("Opcao desconhecida, tente novamente");
138     }
139     printf("\n");
140 } while (!exit);
141 }
142
143 int main() {
144     runMenu();
145     return 0;
146 }
```



```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-06
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-06$ ./ex01

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

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

Elemento para enfileirar no inicio: 1
Enfileirou (1) no inicio

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

Elemento para enfileirar no fim: 2
Enfileirou (2) no fim

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

Elemento para enfileirar no fim: 3
Enfileirou (3) no fim

=====
(1) Criar
(2) Enfileirar inicio
(3) Enfileirar fim
(4) Ver inicio
(5) Ver fim
(6) Desenfileirar inicio
(7) Desenfileirar fim
(8) Imprimir
(9) Destruir
(0) Sair
=====
Operacao: 8

Elementos:
1 2 3

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

Desenfileirou o primeiro elemento

=====
(1) Criar
(2) Enfileirar inicio
(3) Enfileirar fim
(4) Ver inicio
(5) Ver fim
(6) Desenfileirar inicio
(7) Desenfileirar fim
(8) Imprimir
(9) Destruir
(0) Sair
=====
Operacao: 7

Desenfileirou o ultimo elemento

=====
(1) Criar
(2) Enfileirar inicio
(3) Enfileirar fim
(4) Ver inicio
(5) Ver fim
(6) Desenfileirar inicio
(7) Desenfileirar fim
(8) Imprimir
(9) Destruir
(0) Sair
=====
Operacao: 8

Elementos:
2

=====
(1) Criar
(2) Enfileirar inicio
(3) Enfileirar fim
(4) Ver inicio
(5) Ver fim
(6) Desenfileirar inicio
(7) Desenfileirar fim
(8) Imprimir
(9) Destruir
(0) Sair
=====
Operacao: 9

Deque destruido

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

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-06$
```

```
1  #ifndef FPSE_H
2  #define FPSE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO{
8      int info, prio;
9      struct NO* prox;
10 }NO;
11
12 typedef struct NO* Filap;
13
14 NO* alocarNO(){
15     return (NO*) malloc (sizeof(NO));
16 }
17
18 void liberarNO(NO* q){
19     free(q);
20 }
21
22 Filap* criaFila(){
23     Filap* fp;
24     fp = (Filap*) malloc (sizeof(Filap));
25     if(fp != NULL)
26         *fp = NULL;
27     return fp;
28 }
29
30 int estaVazia(Filap* fp){
31     if(fp == NULL) return -1;
32     return ((*fp) == NULL);
33 }
34
35 int inserirPrio(Filap* fp, int elem, int pri){
36     if(fp == NULL) return 0;
37     NO* novo = alocarNO();
38     if(novo == NULL) return 0;
39
40     novo->info = elem;
41     novo->prio = pri;
42
43     if(estaVazia(fp)){
44         novo->prox = *fp;
45         *fp = novo;
46     }else{
47         NO* aux, *ant;
48         ant = NULL;
49         aux = *fp; //Inicio
50         while(aux != NULL && aux->prio >= novo->prio){
51             ant = aux;
52             aux = aux->prox;
53         }
54         if(ant == NULL){
55             novo->prox = *fp;
56             *fp = novo;
57         }else{
58             novo->prox = ant->prox;
59             ant->prox = novo;
60         }
61     }
```

```

61     }
62     return 1;
63 }
64
65 int removeInicio(FilaP* fp){
66     if(fp == NULL) return 0;
67     if(estaVazia(fp)) return 0;
68     NO* aux = *fp;
69     *fp = aux->prox;
70     liberarNO(aux);
71     return 1;
72 }
73
74 int verInicio(FilaP* fp, int* p, int *pri){
75     if(fp == NULL) return 0;
76     if(estaVazia(fp)) return 0;
77     *p = (*fp)->info;
78     *pri = (*fp)->prio;
79     return 1;
80 }
81
82
83 void imprime(FilaP* fp){
84     if(fp == NULL) return;
85     if(estaVazia(fp)){
86         printf("Fila Vazia!\n");
87         return;
88     }
89     NO* aux = *fp;
90     while(aux != NULL){
91         printf("[%d, %d] ", aux->prio, aux->info);
92         aux = aux->prox;
93     }
94     printf("\n");
95 }
96
97 void destroiFila(FilaP** fp){
98     if(*fp != NULL){
99         NO* aux;
100         while((*fp) != NULL){
101             aux = **fp;
102             **fp = (*fp)->prox;
103             liberarNO(aux);
104         }
105         free(*fp);
106         *fp = NULL;
107     }
108 }
109
110 int tamanhoFila (FilaP* fp) {
111     if (fp == NULL) return -1;
112
113     int t = 0;
114     NO* i = *fp;
115     while (i != NULL) {
116         t++;
117         i = i->prox;
118     }
119
120     return t;
121 }
122
123 #endif

```

roteiro-06/ex02-02.h

```
1  /*----- File: FPHeap.h -----+
2  |Fila de Prioridades com Heap Binaria |
3  | |
4  | |
5  | Implementado por Guilherme C. Pena em 26/09/2023 |
6  +-----+ */
7
8  #ifndef FPHEAP_H
9  #define FPHEAP_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 #define MAX 100
15
16 typedef struct NO{
17     int info, prio;
18 }NO;
19
20 typedef struct{
21     int qtd;
22     NO dados[MAX];
23 }FilaP;
24
25 FilaP* criaFila(){
26     FilaP* fp;
27     fp = (FilaP*) malloc (sizeof(FilaP));
28     if(fp != NULL)
29         fp->qtd = 0;
30     return fp;
31 }
32
33 void destroiFila(FilaP** fp){
34     if(*fp != NULL) {
35         free(*fp);
36         *fp = NULL;
37     }
38 }
39
40 int tamanhoFila(FilaP *fp){
41     if(fp == NULL) return -1;
42     return fp->qtd;
43 }
44
45 int estaCheia(FilaP *fp){
46     if(fp == NULL) return -1;
47     return (fp->qtd == MAX);
48 }
49
50 int estaVazia(FilaP *fp){
51     if(fp == NULL) return -1;
52     return (fp->qtd == 0);
53 }
54
```

```

55 void trocaNO(NO* a, NO* b){
56     NO temp;
57     temp.info = a->info;
58     temp.prio = a->prio;
59     a->info = b->info;
60     a->prio = b->prio;
61     b->info = temp.info;
62     b->prio = temp.prio;
63 }
64
65 void ajustaHeapInsere(FilaP* fp, int filho){
66     NO temp;
67     int pai = (filho-1)/2;
68     int prioPai = fp->dados[pai].prio;
69     int prioFilho = fp->dados[filho].prio;
70     while(filho > 0 && prioPai < prioFilho){
71         trocaNO(&fp->dados[filho], &fp->dados[pai]);
72         filho = pai;
73         pai = (pai-1)/2;
74         prioPai = fp->dados[pai].prio;
75         prioFilho = fp->dados[filho].prio;
76     }
77 }
78
79 int inserirPrio(FilaP* fp, int elem, int pri){
80     if(fp == NULL) return 0;
81     if(estaCheia(fp)) return 0;
82     fp->dados[fp->qtd].info = elem;
83     fp->dados[fp->qtd].prio = pri;
84     ajustaHeapInsere(fp, fp->qtd);
85     fp->qtd++;
86     return 1;
87 }
88
89 void ajustaHeapRemove(FilaP* fp, int pai){
90     NO temp;
91     int filho = 2*pai + 1;
92     while(filho < fp->qtd){
93         if(filho < fp->qtd-1)
94             if(fp->dados[filho].prio < fp->dados[filho+1].prio)
95                 filho++;
96
97         if(fp->dados[pai].prio > fp->dados[filho].prio)
98             break;
99
100         trocaNO(&fp->dados[pai], &fp->dados[filho]);
101         pai = filho;
102         filho = 2*pai + 1;
103     }
104 }
105
106 int removeInicio(FilaP* fp){
107     if(fp == NULL) return 0;
108     if(estaVazia(fp)) return 0;
109
110     fp->qtd--;
111     fp->dados[0].info = fp->dados[fp->qtd].info;

```

```
112     fp->dados[0].prio = fp->dados[fp->qtd].prio;
113     ajustaHeapRemove(fp, 0);
114     return 1;
115 }
116
117 int verInicio(FilaP* fp, int* valor, int* pri){
118     if(fp == NULL) return 0;
119     if(estaVazia(fp)) return 0;
120     *valor = fp->dados[0].info;
121     *pri = fp->dados[0].prio;
122     return 1;
123 }
124
125 void imprime(FilaP* fp){
126     if(fp == NULL) return;
127     if(estaVazia(fp)){
128         printf("Fila Vazia!\n");
129         return;
130     }
131     printf("Elementos:\n");
132     int i;
133     for(i=0; i<fp->qtd; i++)
134         printf("[%d, %d] (%d) -- ", fp->dados[i].prio, fp->dados[i].info, i);
135     printf("\n");
136 }
137
138 #endif
```

roteiro-06/ex02.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*
5   *
6   *  O ARQUIVO FUNCIONA PARA AS DUAS IMPLEMENTACOES DE FILA DE PRIORIDADES
7   *  BASTA IMPORTAR APENAS A BIBLIOTECA DESEJADA
8   *
9   *  EX02-01 = FILA DE PRIORIDADES SIMPLEMENTE ENCADEADA
10  *  EX02-02 = HEAP
11  *
12  */
13
14  #include "ex02-01.h"
15  // #include "ex02-02.h"
16
17  enum {
18      EXIT = 0,
19      CREATE,
20      QUEUE,
21      START,
22      DEQUEUE,
23      PRINT,
24      SIZE,
25      DESTROY
26  } Options;
27
28  int getOption() {
29      int option;
30
31      printf("\n=====\\n");
32      printf("(\\d) Criar\\n", CREATE);
33      printf("(\\d) Enfileirar\\n", QUEUE);
34      printf("(\\d) Ver inicio\\n", START);
35      printf("(\\d) Desenfileirar\\n", DEQUEUE);
36      printf("(\\d) Imprimir\\n", PRINT);
37      printf("(\\d) Tamanho\\n", SIZE);
38      printf("(\\d) Destruir\\n", DESTROY);
39      printf("(\\d) Sair\\n", EXIT);
40      printf("=====\\n");
41      printf("Operacao: ");
42
43      scanf("%d", &option);
44      printf("\\n");
45
46      return option;
47  }
48
49  int runMenu() {
50      FilaP* fila = NULL;
51      int exit = 0, item, pri;
52  }
```

```

53     do {
54         switch (getOption()) {
55             case CREATE:
56                 if (fila != NULL) {
57                     destroiFila(&fila);
58                     printf("Fila resetada");
59                 }
60                 fila = criaFila();
61                 break;
62
63             case QUEUE:
64                 printf("Elemento para enfileirar: ");
65                 scanf("%d", &item);
66                 printf("Prioridade: ");
67                 scanf("%d", &pri);
68                 if (inserirPrio(fila, item, pri)) {
69                     printf("Enfileirou (%d)", item);
70                 } else {
71                     printf("Nao foi possivel enfileirar (%d)", item);
72                 }
73                 break;
74
75             case START:
76                 if (verInicio(fila, &item, &pri)) {
77                     printf("Inicio da fila = %d (Prioridade = %d)\n", item,
78 pri);
79                 } else {
80                     printf("Nao foi possivel ver o inicio da fila");
81                 }
82                 break;
83
84             case DEQUEUE:
85                 if (removeInicio(fila)) {
86                     printf("Desenfileirou o primeiro elemento");
87                 } else {
88                     printf("Nao foi possivel desenfileirar");
89                 }
90                 break;
91
92             case PRINT:
93                 imprime(fila);
94                 break;
95
96             case SIZE:
97                 printf("Tamanho da fila = %d", tamanhoFila(fila));
98                 break;
99
100             case DESTROY:
101                 destroiFila(&fila);
102                 printf("Fila destruida");
103                 break;
104
105             case EXIT:
106                 if (fila != NULL) {
107                     destroiFila(&fila);
108                 }
109             }
110         }
111     }

```



```
108         printf("Programa encerrado");
109         exit = 1;
110         break;
111
112         default:
113             printf("Opcao desconhecida, tente novamente");
114     }
115     printf("\n");
116 } while (!exit);
117 }
118
119 int main() {
120     runMenu();
121     return 0;
122 }
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-06
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-06$ ./ex02

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

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

Elemento para enfileirar: 23
Prioridade: 5
Enfileirou (23)

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

Elemento para enfileirar: 19
Prioridade: 10
Enfileirou (19)

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

Elemento para enfileirar: 15
Prioridade: 7
Enfileirou (15)

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

[10, 19] [7, 15] [5, 23]

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

Desenfileirou o primeiro elemento

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

Desenfileirou o primeiro elemento

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

Tamanho da fila = 1

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

[5, 23]

=====
(1) Criar
(2) Enfileirar
(3) Ver inicio
(4) Desenfileirar
(5) Imprimir
(6) Tamanho
(7) Destruir
(0) Sair
=====
Operacao: 7

Fila destruida

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

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-06$
```