roteiro-06/ex01-01.h

```c
#ifndef DEQUE_H
#define DEQUE_H

#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct {
    int qtd, ini, fim;
    int dados[MAX];
} Deque;

Deque* criaDeque() {
    Deque* dq;
    dq = (Deque*)malloc(sizeof(Deque));
    if (dq != NULL) {
        dq->qtd = dq->ini = dq->fim = 0;
    }
    return dq;
}

void destroiDeque(Deque** dq) {
    if (*dq != NULL) {
        free(*dq);
        *dq = NULL;
    }
}

int tamanhoDeque(Deque* dq) {
    if (dq == NULL)
        return -1;
    return dq->qtd;
}

int estaCheio(Deque* dq) {
    if (dq == NULL)
        return -1;
    return (dq->qtd == MAX);
}

int estaVazio(Deque* dq) {
    if (dq == NULL)
        return -1;
    return (dq->qtd == 0);
}

int insereInicio(Deque* dq, int elem) {
    if (dq == NULL) return 0;
    if (estaCheio(dq)) return 0;
    dq->ini = (dq->ini - 1 < 0 ? MAX - 1 : dq->ini - 1);
    dq->dados[dq->ini] = elem;
    dq->qtd++;
    return 1;
}

```

```c
int insereFim(Deque* dq, int elem) {
    if (dq == NULL) return 0;
    if (estaCheio(dq)) return 0;
    dq->dados[dq->fim] = elem;
    dq->fim = (dq->fim + 1) % MAX;
    dq->qtd++;
    return 1;
}

int removeInicio(Deque* dq) {
    if (dq == NULL) return 0;
    if (estaVazio(dq)) return 0;
    dq->ini = (dq->ini + 1) % MAX;
    dq->qtd--;
    return 1;
}

int removeFim(Deque* dq) {
    if (dq == NULL) return 0;
    if (estaVazio(dq)) return 0;
    dq->fim = (dq->fim - 1 < 0 ? MAX - 1 : dq->fim - 1);
    dq->qtd--;
    return 1;
}

int verInicio(Deque* dq, int* p) {
    if (dq == NULL) return 0;
    if (estaVazio(dq)) return 0;
    *p = dq->dados[dq->ini];
    return 1;
}

int verFim(Deque* dq, int* p) {
    if (dq == NULL) return 0;
    if (estaVazio(dq)) return 0;
    int i = (dq->fim - 1 < 0 ? MAX - 1 : dq->fim - 1);
    *p = dq->dados[i];
    return 1;
}

void imprime(Deque* dq) {
    if (dq == NULL) return;
    if (estaVazio(dq)) {
        printf("Deque Vazio!\n");
        return;
    }
    int i = dq->ini;
    printf("Elementos: \n");
    do {
        printf("%d ", dq->dados[i]);
        i = (i + 1) % MAX;
    } while (i != dq->fim);
    // Usar do..while garante a impressao de todos elementos
    // mesmo com a Deque cheia
    printf("\n");
}

#endif
```

roteiro-06/ex01-02.h

```c
#ifndef DDE_H
#define DDE_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO{
    int info;
    struct NO* prox;
    struct NO* ant;
}NO;

typedef struct{
    int qtd;
    struct NO* ini;
    struct NO* fim;
}Deque;


NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

Deque* criaDeque(){
    Deque* dq;
    dq = (Deque*) malloc (sizeof(Deque));
    if(dq != NULL){
        dq->qtd = 0;
        dq->ini = NULL;
        dq->fim = NULL;
    }
    return dq;
}

void destroiDeque(Deque **dq){
    if(*dq != NULL){
        NO* aux;
        while((*dq)->ini != NULL){
            aux = (*dq)->ini;
            (*dq)->ini = (*dq)->ini->prox;
            liberarNO(aux);
        }
        free(*dq);
        *dq = NULL;
    }
}

int tamanhoDeque(Deque *dq){
    if(dq == NULL)
        return -1;
    return dq->qtd;
}
```

```
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){
```

```
    if(dq == NULL) return 0;
    if(estaVazio(dq)) return 0;
    NO* aux = dq->fim;
    if(dq->ini == dq->fim){
        dq->ini = dq->fim = NULL;
    }else{
        dq->fim = dq->fim->ant;
        dq->ini->prox = NULL;
    }
    liberarNO(aux);
    dq->qtd--;
    return 1;
}

int verInicio(Deque* dq, int* p){
  if(dq == NULL) return 0;
  if(estaVazio(dq)) return 0;
  *p = dq->ini->info;
  return 1;
}

int verFim(Deque* dq, int* p){
  if(dq == NULL) return 0;
  if(estaVazio(dq)) return 0;
  *p = dq->fim->info;
  return 1;
}

void imprime(Deque* dq){
  if(dq == NULL) return;
  if(estaVazio(dq)){
    printf("Deque Vazio!\n");
    return;
  }
  NO* aux = dq->ini;
  printf("Elementos:\n");
  while(aux != NULL){
    printf("%d ", aux->info);
    aux = aux->prox;
  }
  printf("\n");
}

#endif
```

roteiro-06/ex01.c

```c
#include <stdio.h>
#include <stdlib.h>

/*
 *
 *  O ARQUIVO FUNCIONA PARA AS DUAS IMPLEMENTACOES DE DEQUE
 *  BASTA IMPORTAR APENAS A BIBLIOTECA DESEJADA
 *
 *  EX01-01 = DEQUE SEQUENCIAL ESTATICO
 *  EX01-02 = DEQUE SIMPLESMENTE ENCADEADO
 *
 */

// #include "ex01-01.h"
#include "ex01-02.h"

enum {
    EXIT = 0,
    CREATE,
    QUEUE_START,
    QUEUE_END,
    START,
    END,
    DEQUEUE_START,
    DEQUEUE_END,
    PRINT,
    DESTROY
} Options;

int getOption() {
    int option;

    printf("\n====================\n");
    printf("(%d) Criar\n", CREATE);
    printf("(%d) Enfileirar inicio\n", QUEUE_START);
    printf("(%d) Enfileirar fim\n", QUEUE_END);
    printf("(%d) Ver inicio\n", START);
    printf("(%d) Ver fim\n", END);
    printf("(%d) Desenfileirar inicio\n", DEQUEUE_START);
    printf("(%d) Desenfileirar fim\n", DEQUEUE_END);
    printf("(%d) Imprimir\n", PRINT);
    printf("(%d) Destruir\n", DESTROY);
    printf("(%d) Sair\n", EXIT);
    printf("====================\n");
    printf("Operacao: ");

    scanf("%d", &option);
    printf("\n");

    return option;
}

int runMenu() {
    Deque* deque = NULL;
    int exit = 0, item;

```

```c
    do {
        switch (getOption()) {
            case CREATE:
                if (deque != NULL) {
                    destroiDeque(&deque);
                    printf("Deque resetado");
                }
                deque = criaDeque();
                break;

            case QUEUE_START:
                printf("Elemento para enfileirar no inicio: ");
                scanf("%d", &item);
                if (insereInicio(deque, item)) {
                    printf("Enfileirou (%d) no inicio", item);
                } else {
                    printf("Nao foi possivel enfileirar (%d)", item);
                }
                break;

            case QUEUE_END:
                printf("Elemento para enfileirar no fim: ");
                scanf("%d", &item);
                if (insereFim(deque, item)) {
                    printf("Enfileirou (%d) no fim", item);
                } else {
                    printf("Nao foi possivel enfileirar (%d)", item);
                }
                break;

            case START:
                if (verInicio(deque, &item)) {
                    printf("Inicio do deque = %d\n", item);
                } else {
                    printf("Nao foi possivel ver o inicio do deque");
                }
                break;

            case END:
                if (verFim(deque, &item)) {
                    printf("Fim do deque = %d\n", item);
                } else {
                    printf("Nao foi possivel ver o fim do deque");
                }
                break;

            case DEQUEUE_START:
                if (removeInicio(deque)) {
                    printf("Desenfileirou o primeiro elemento");
                } else {
                    printf("Nao foi possivel desenfileirar");
                }
                break;

            case DEQUEUE_END:
                if (removeFim(deque)) {
                    printf("Desenfileirou o ultimo elemento");
                } else {
                    printf("Nao foi possivel desenfileirar");
```

```
                }
                break;

            case PRINT:
                imprime(deque);
                break;

            case DESTROY:
                destroiDeque(&deque);
                printf("Deque destruido");
                break;

            case EXIT:
                if (deque != NULL) {
                    destroiDeque(&deque);
                }
                printf("Programa encerrado");
                exit = 1;
                break;

            default:
                printf("Opcao desconhecida, tente novamente");
        }
        printf("\n");
    } while (!exit);
}

int main() {
    runMenu();
    return 0;
}
```

```
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 destruído

====================
(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$
```

roteiro-06/ex02-01.h

```h
#ifndef FPSE_H
#define FPSE_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO{
    int info, prio;
    struct NO* prox;
}NO;

typedef struct NO* FilaP;

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

FilaP* criaFila(){
    FilaP* fp;
    fp = (FilaP*) malloc (sizeof(FilaP));
    if(fp != NULL)
        *fp = NULL;
    return fp;
}

int estaVazia(FilaP* fp){
    if(fp == NULL) return -1;
    return ((*fp) == NULL);
}

int inserirPrio(FilaP* fp, int elem, int pri){
    if(fp == NULL) return 0;
    NO* novo = alocarNO();
    if(novo == NULL) return 0;

    novo->info = elem;
    novo->prio = pri;

    if(estaVazia(fp)){
        novo->prox = *fp;
        *fp = novo;
    }else{
        NO* aux, *ant;
        ant = NULL;
        aux = *fp;//Inicio
        while(aux != NULL && aux->prio >= novo->prio){
            ant = aux;
            aux = aux->prox;
        }
        if(ant == NULL){
            novo->prox = *fp;
            *fp = novo;
        }else{
            novo->prox = ant->prox;
            ant->prox = novo;
        }
```

```
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

```c
/*----------------- File: FPHeap.h ---------------------+
|Fila de Prioridades com Heap Binaria                   |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 26/09/2023      |
+-------------------------------------------------------+ */

#ifndef FPHEAP_H
#define FPHEAP_H

#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct NO{
    int info, prio;
}NO;

typedef struct{
    int qtd;
    NO dados[MAX];
}FilaP;

FilaP* criaFila(){
    FilaP* fp;
    fp = (FilaP*) malloc (sizeof(FilaP));
    if(fp != NULL)
        fp->qtd = 0;
    return fp;
}

void destroiFila(FilaP** fp){
    if(*fp != NULL) {
        free(*fp);
        *fp = NULL;
    }
}

int tamanhoFila(FilaP *fp){
  if(fp == NULL) return -1;
  return fp->qtd;
}

int estaCheia(FilaP *fp){
  if(fp == NULL) return -1;
  return (fp->qtd == MAX);
}

int estaVazia(FilaP *fp){
  if(fp == NULL) return -1;
  return (fp->qtd == 0);
}

```

```
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

```c
#include <stdio.h>
#include <stdlib.h>

/*
 *
 *  O ARQUIVO FUNCIONA PARA AS DUAS IMPLEMENTACOES DE FILA DE PRIORIDADES
 *  BASTA IMPORTAR APENAS A BIBLIOTECA DESEJADA
 *
 *  EX02-01 = FILA DE PRIORIDADES SIMPLESMENTE ENCADEADA
 *  EX02-02 = HEAP
 *
 */

#include "ex02-01.h"
// #include "ex02-02.h"

enum {
    EXIT = 0,
    CREATE,
    QUEUE,
    START,
    DEQUEUE,
    PRINT,
    SIZE,
    DESTROY
} Options;

int getOption() {
    int option;

    printf("\n====================\n");
    printf("(%d) Criar\n", CREATE);
    printf("(%d) Enfileirar\n", QUEUE);
    printf("(%d) Ver inicio\n", START);
    printf("(%d) Desenfileirar\n", DEQUEUE);
    printf("(%d) Imprimir\n", PRINT);
    printf("(%d) Tamanho\n", SIZE);
    printf("(%d) Destruir\n", DESTROY);
    printf("(%d) Sair\n", EXIT);
    printf("====================\n");
    printf("Operacao: ");

    scanf("%d", &option);
    printf("\n");

    return option;
}

int runMenu() {
    FilaP* fila = NULL;
    int exit = 0, item, pri;

```

```c
    do {
        switch (getOption()) {
            case CREATE:
                if (fila != NULL) {
                    destroiFila(&fila);
                    printf("Fila resetada");
                }
                fila = criaFila();
                break;

            case QUEUE:
                printf("Elemento para enfileirar: ");
                scanf("%d", &item);
                printf("Prioridade: ");
                scanf("%d", &pri);
                if (inserirPrio(fila, item, pri)) {
                    printf("Enfileirou (%d)", item);
                } else {
                    printf("Nao foi possivel enfileirar (%d)", item);
                }
                break;

            case START:
                if (verInicio(fila, &item, &pri)) {
                    printf("Inicio da fila = %d (Prioridade = %d)\n", item,
pri);
                } else {
                    printf("Nao foi possivel ver o inicio da fila");
                }
                break;

            case DEQUEUE:
                if (removeInicio(fila)) {
                    printf("Desenfileirou o primeiro elemento");
                } else {
                    printf("Nao foi possivel desenfileirar");
                }
                break;

            case PRINT:
                imprime(fila);
                break;

            case SIZE:
                printf("Tamanho da fila = %d", tamanhoFila(fila));
                break;

            case DESTROY:
                destroiFila(&fila);
                printf("Fila destruida");
                break;

            case EXIT:
                if (fila != NULL) {
                    destroiFila(&fila);
                }
```

```c
                printf("Programa encerrado");
                exit = 1;
                break;

            default:
                printf("Opcao desconhecida, tente novamente");
        }
        printf("\n");
    } while (!exit);
}

int main() {
    runMenu();
    return 0;
}
```

```
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$
```