**3 - 1** Implemente uma fila com vetor, conforme a interface dada *fila.h*:

1. O TAD TFila, com vetor de 100 elementos e as variáveis com a posições do início e fim;
2. Operação de criar;
3. Operação de enfileirar;
4. Operação de desenfileirar;
5. Operação que retorna tamanho;
6. Operação que retorna posição do início.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include "queue.h"  struct Queue {  int vector[5];  int start, final, quantity;  };  TQueue \*neww() {  TQueue\* queue = (TQueue\*)malloc(sizeof(TQueue));  queue -> start = -1;  queue -> final = -1;  queue -> quantity = 0;  return queue;  }  void enqueue(TQueue \*queue, int value) {  if(!((queue -> final + 1) % 5 == queue -> start)) {  if(queue -> start == -1) queue -> start = 0;  queue -> final = (queue -> final + 1) % 5;  queue -> vector[queue -> final] = value;  queue -> quantity++;  }  }  int dequeue(TQueue \*queue) {  int queuing = -1;  if(queue -> start > -1) {  queuing = queue -> vector[queue -> start];  if(queue -> start == queue -> final) {  queue -> start = queue -> final = -1;  } else {  queue -> start = (queue -> start + 1) % 5;  queue -> quantity--;  }  }  return queuing;  }  int queueSize(TQueue \*queue) {  return queue -> quantity;  }  int queueFrontPosition(TQueue \*queue) {  if(queue -> start > -1) {  return queue -> vector[frontPosition(queue)];  }  } |

**3 - 2** Inclua na interface fila.h e implemente as operações:

1. “vazia()”, a qual retorna 0 se a fila não estiver vazia e 1 em caso contrário.
2. “cheia()”, a qual retorna 0 se a fila não estiver cheia e 1 em caso contrário.

|  |
| --- |
| //queue.h  int empty(TQueue\* queue);  int full(TQueue\* queue);  // queue.c  int empty(TQueue \*queue) {  if(queue -> start == -1) {  return 0;  } else {  return 1;  }  }  int full(TQueue \*queue) {  if(((queue -> final + 1) % 5 == queue -> start)) {  return 0;  } else {  return 1;  }  } |

**3 - 3** Usando as operações do TAD Fila, implemente funções para imprimir todos os elementos da Fila:

1. Esvaziando a fila;
2. Sem esvaziar a fila;

|  |
| --- |
| void printEmptyingValue(TQueue \*queue) {  int i;  int sizeQueue = queueSize(queue);  int valueDequeue;  for(i = 0; i < sizeQueue; i++) {  valueDequeue = dequeue(queue);  printf("\n%i\n", valueDequeue);  }  }  void printValueWithoutEmptying(TQueue \*queue) {  int i;  int sizeQueue = queueSize(queue);  int valueDequeue;  TQueue\* queueAux = neww();  for(i = 0; i < sizeQueue; i++) {  valueDequeue = dequeue(queue);  printf("\n%i\n", valueDequeue);  enqueue(queueAux, valueDequeue);  enqueue(queue, dequeue(queueAux));  }  } |

**3 - 4** Desenvolva uma função que receba uma TFila e um valor n como parâmetros e remova a primeira ocorrência de n da fila, se houver. A função poderá utilizar uma fila auxiliar e só poderá utilizar as operações de fila para manipular seus elementos.

|  |
| --- |
| int removeValue(TQueue \*queue, int input) {  if(queue == empty(queue)) {  printf("Fila esta vazia...");  return;  }    int i = queueSize(queue) - 1;  int j;  for(j = 0; queueFrontPosition(queue) != input; i--, j++) {  enqueue(queue, dequeue(queue));  if(j == queueSize(queue) - 1) {  return;  }  }    dequeue(queue);  for(i = i; i > 0; i--) {  enqueue(queue, dequeue(queue));  }  } |

**3 - 5** Desenvolva uma função que receba uma TFila e retorne uma fila com os seus valores invertidos. A função poderá utilizar uma pilha auxiliar e só poderá utilizar as operações dos TADs para manipular seus elementos.

|  |
| --- |
| void invert(TQueue \*queue) {  int i, size = queueSize(queue);  int vector[size];  for(i = 0; !empty(queue); i++) {  vector[i] = dequeue(queue);  }    for(i = size; i >= 0; i--) {  enqueue(queue, vector[i]);  }  } |

**3 - 6** Implemente as operações de suporte a filas de prioridades:

1. Corrige subindo;
2. Corrige descendo.

|  |
| --- |
| void correctUp(TQueuePriority \*queue) {  int i = queue -> size;  while(i >= 2 && queue -> vector[i/2] < queue -> vector[i]) {  swap(queue, i/2, i);  i /= 2;  }  }  void correctDown(TQueuePriority \*queue) {  int i = 1;  while(2 \* i <= queue -> size) {  int j = 2 \* i;  if(j < queue -> size && queue -> vector[j] < queue -> vector[j + 1]) {  j++;  }    if(queue -> vector[i] >= queue -> vector[j]) {  i = queue -> size;  } else {  swap(queue, i, j);  i = j;  }  }  } |

**3 - 7** Implemente uma fila com vetor, conforme a interface dada *filapri.h*:

1. O TAD TFilaPri, com vetor de 100 elementos e uma variável n com a quantidade de elementos na fila;
2. Operação de encontrarMax;
3. Operação de extrairMax;
4. Operação de inserir;

|  |
| --- |
| int findBigger(TQueuePriority \*queue) {  int i, biggerValue;  for (i = queue -> ini, biggerValue = queue -> vector[i]; i != queue -> fim; i = i == TAM-1 ? 0 : i+1) {  if(queue -> vector[i] > biggerValue) {  biggerValue = queue -> vector[i];  }  }  return biggerValue;  }  int extractMax(TQueuePriority \*queue){  if(queue -> quantity > 0) {  int max = queue -> vector[1];  queue -> vector[1] = queue -> vector[queue -> quantity];  queue -> quantity = queue -> quantity -1;  correctDown(queue);  return max;  }  }  void insert(TQueuePriority \*queue, int value) {  if(queue -> quantity < TAM) {  queue -> quantity = queue -> qtd +1;  queue -> vetor[queue -> quantity] = value;  corrigirSubindo(queue);  }  } |